



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Jenkins Essentials

Continuous Integration – setting up the stage for a DevOps culture

Mitesh Soni

[PACKT] open source*
PUBLISHING community experience distilled

Jenkins Essentials

Continuous Integration – setting up the stage for a DevOps culture

Mitesh Soni



BIRMINGHAM - MUMBAI

Jenkins Essentials

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2015

Production reference: 1220715

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78355-347-1

www.packtpub.com

Credits

Author

Mitesh Soni

Project Coordinator

Bijal Patel

Reviewers

Anthony Dahanne

Michael Peacock

Devin Young

Proofreader

Safis Editing

Indexer

Hemangini Bari

Commissioning Editor

Nadeem N. Bagban

Graphics

Disha Haria

Acquisition Editors

Indrajit Das

Rebecca Youé

Production Coordinator

Shantanu N. Zagade

Content Development Editor

Shubhangi Dhamgaye

Cover Work

Shantanu N. Zagade

Technical Editor

Mitali Somaiya

Copy Editors

Janbal Dharmaraj

Kevin McGowan

About the Author

Mitesh Soni is currently working as a technical specialist at IGATE. He is an SCJP, SCWCD, and VCP. While he has interest in DevOps and Cloud computing, his real passion is to play with kids, play with his camera, and capture photographs at Indroda Park. He loves programming in Java, and he finds design patterns fascinating. He lives in the capital of Mahatma Gandhi's home state. He loves to spend time alone and loves walking at Punit Van. He believes that without a sense of urgency, desire loses its value. He has earlier authored *Learning Chef* by Packt Publishing (<https://www.packtpub.com/networking-and-servers/learning-chef>).

I want to say thanks and express my gratitude for everything I've been blessed with. I would like to thank Jigisha-Nitesh, dada-dadi, my teachers, friends, and family members who have always supported me. Special thanks to Vishwajit for encouraging me to work on Jenkins. I would like to thank Jyoti Namjoshi for encouraging me to write articles and for her guidance and valuable support in what I do. I would also like to thank the IGATE senior management for providing opportunities to explore latest technology trends and work on them extensively.

About the Reviewers

Anthony Dahanne has been a Java software developer for 10 years. His favorite topics include web apps, building tools, continuous integration and, of course, core Java development.

Passionate in delivering valuable software, you'll often see Anthony at start up events or user groups in Montreal.

Working for Terracotta, he is part of the management and monitoring team that makes the products easily monitorable with REST APIs and builds a nice UI.

He is the author of *Instant Spring for Android Starter* by Packt Publishing.

I'd like to thank my family for their support and patience while I'm busy discovering new software technologies!

Michael Peacock is an experienced software developer and team lead from Newcastle, UK, with a degree in software engineering from Durham University.

After spending a number of years running his own web agency and subsequently working directly for a number of software start-ups, he now runs his own software development agency, working on a range of projects for an array of different clients.

He is the author of *Creating Development Environments with Vagrant*, *PHP 5 Social Networking*, *PHP 5 E-Commerce Development*, *Drupal 7 Social Networking*, *Selling Online with Drupal e-Commerce*, and *Building Websites with TYPO3* all by Packt Publishing. Other books he has been involved with include *Advanced API Security*, *Mobile Web Development*, *Jenkins Continuous Integration Cookbook*, and *Drupal for Education and E-Learning*, for which he acted as a technical reviewer.

He has also presented at a number of user groups and technical conferences, including PHP UK Conference, Dutch PHP Conference, ConFoo, PHPNE, PHPNW, and CouldConnect Santa Clara.

You can follow him on Twitter at [@michaelpacock](#) or find out more about him through his website at [www.michaelpacock.co.uk](#).

I'd like to thank the team at Packt Publishing for their help and support.

Devin Young graduated with a BS in sports management from The Ohio State University and somehow wound up working as a software engineer shortly afterwards. He specializes in DevOps and is particularly fond of automation and real-time applications. He grew up as a competitive jump roper, leading him to create the mobile app RopeRacer, which was launched on iOS in March 2015. The app has become a success in the world of jump rope and is now used in tournaments around the United States.

I would like to thank my family and coworkers for always putting up with me and my soon-to-be wife, Christen, for letting me be nerdy.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	v
Chapter 1: Exploring Jenkins	1
Introduction to Jenkins and its features	2
Features	3
Installation of Jenkins on Windows and CentOS	4
Installing Jenkins on Windows	4
Installation of Jenkins on CentOS	7
Installation of Jenkins as a web application	9
A jump-start tour of the Jenkins dashboard	10
How to change configuration settings in Jenkins	12
What is the deployment pipeline?	15
Self-test questions	16
Summary	17
Chapter 2: Installation and Configuration of Code Repository and Build Tools	19
An overview of a build in Jenkins and its requirements	20
Installing Java and configuring environment variables	20
Configure environment variables	22
Installing, configuring, and operating SVN on CentOS and Windows	23
Installing SVN on CentOS	23
Configuring SVN	24
SVN operations	26
Import a directory into SVN	26
Check out from SVN	27
VisualSVN Server on Windows	28
Integrating Eclipse with code repositories	36
Installing and configuring Ant	38
Installing Maven	39

Configuring Ant, Maven, and JDK in Jenkins	40
Installing and configuring Git	41
Creating a new build job in Jenkins with Git	44
Self-test questions	48
Summary	48
Chapter 3: Integration of Jenkins, SVN, and Build Tools	49
Creating and configuring a build job for a Java application with Ant	50
Dashboard View Plugin	50
Creating and configuring a build job for a Java application	52
Creating and configuring a build job for a Java application with Maven	58
Build execution with test cases	62
Self-test questions	66
Summary	66
Chapter 4: Implementing Automated Deployment	67
An overview of continuous delivery and continuous deployment	68
Installing Tomcat	68
Deploying a war file from Jenkins to Tomcat	70
Self-test questions	76
Summary	77
Chapter 5: Hosted Jenkins	79
Exploring Jenkins in OpenShift PaaS	80
Exploring Jenkins in the Cloud – CloudBees	84
An overview of CloudBees Enterprise Plugins	94
Workflow Plugin	94
Checkpoints Plugin	94
Role-based Access Control Plugin	95
High Availability Plugin	95
VMware ESXi/vSphere Auto-Scaling Plugin	95
Jenkins case studies from CloudBees	96
Apache jclouds	96
Challenge	96
Solution	96
Benefits	97

Global Bank	97
Challenge	97
Solution	97
Benefits	98
Service-Flow	98
Challenge	98
Solution	98
Benefits	98
Self-test questions	99
Summary	99
Chapter 6: Managing Code Quality and Notifications	101
Integration with Sonar	101
Exploring Static Code Analysis Plugins	110
Checkstyle Plugin	110
FindBugs Plugin	111
Compiler Warnings Plugin	111
DRY Plugin	113
PMD Plugin	113
Task Scanner Plugin	113
CCM Plugin	113
Android Lint Plugin	113
OWASP Dependency-Check Plugin	114
E-mail notifications on build status	114
Self-test questions	115
Summary	115
Chapter 7: Managing and Monitoring Jenkins	117
Managing Jenkins master and slave nodes	118
Jenkins monitoring with JavaMelody	123
Managing disk usage	125
Build monitoring with Build Monitor Plugin	127
Managing access control and authorization	129
Maintaining roles and project-based security	133
Audit Trail Plugin – an overview and usage	137
Self-test questions	139
Summary	139

Chapter 8: Beyond Basics of Jenkins – Leveraging "Must-have" Plugins	141
Extended Email Plugin	142
Workspace cleanup Plugin	144
Pre-scm-buildstep Plugin	146
Conditional BuildStep Plugin	148
EnvInject Plugin	150
Build Pipeline Plugin	152
Self-test questions	156
Summary	157
Index	159

Preface

DevOps is a buzz word in 2015 and will be for the coming years as per market trends by various research firms. In DevOps culture, business owners, development teams, operations teams, and QA teams collaborate to deliver outcome in a continuous and effective manner. It enables the organizations to more quickly grab opportunities and reduce the time taken to include customer feedback into new feature development or innovation. The end goal of DevOps is to reduce the time between the initial concept and the end result of the concept in the form of production ready applications.

DevOps targets application delivery, new feature development, bug fixing, testing, and maintenance releases. It improves efficiency, security, reliability, predictability, and faster development and deployment cycles. It covers all SDLC phases from development, test, operations, and release.

Continuous integration (CI) and continuous delivery (CD) are a significant part of the DevOps culture. Jenkins is a fully featured technology platform that enables users to implement CI and CD. This helps users to deliver better applications by automating the application delivery life cycle. CI includes automation of build, test and package processes. CD includes the application delivery pipeline across different environments. Jenkins enables the user to utilize continuous integration services for software development in an agile environment. Continuous integration systems are a vital part of the agile team because they help enforce the principles of agile development. Continuous Integration is a significant part of the DevOps culture, and hence, many open source and commercial tools for continuous delivery utilize Jenkins or provide integration points. Jenkins enables agile teams to focus on work and innovations by automating the build, artifact management, and deployment processes, rather than worrying about manual processes. It can be used to build freestyle software projects based on Apache Ant and Maven 2 / Maven 3 projects. It can also execute Windows batch commands and shell scripts.

There are a number of ways to install Jenkins, and it can be used across different platforms such as Windows and Linux. Jenkins is available in the form of native packages of Windows, FreeBSD, OpenBSD, Red Hat, Fedora, CentOS, Ubuntu, Debian, Mac OS X, openSUSE, Solaris, OpenIndiana, Gentoo, or in the form of WAR file. The quickest and easiest way to use Jenkins is to use the WAR file. It can be easily customized with the use of plugins. There are different kinds of plugins available to customize Jenkins based on specific needs. Categories of plugins include source code management (that is, Git Plugin, CVS Plugin, and Bazaar Plugin), build triggers (that is, Accelerated Build Now Plugin and Build Flow Plugin), build reports (that is, CodeScanner Plugin and Disk Usage Plugin), authentication and user management (that is, Active Directory Plugin and Github OAuth Plugin), cluster management and distributed build (that is, Amazon EC2 Plugin and Azure Slave Plugin), and so on.

Jenkins is very popular among its users as it allows them to manage and control phases such as build, test, package, and static code analysis. It has won InfoWorld Bossies Award, 2011; O'Reilly Open Source Award, 2011; ALM&SCM; and so on. The main users of Jenkins are NASA, LinkedIn, eBay, and Mozilla Foundation.

The following are some features that make Jenkins very popular:

- An open source tool with a web-based GUI.
- A Java-based continuous build system—easy to write plugins.
- Highly configurable tool—a plugin-based architecture that provides support to many technology, repositories, build tools, and test tools.
- The Jenkins user community is large and active. It has more than 1,000 open source plugins.
- This supports CI for .Net, iOS, Android, and Ruby development.
- This supports common SCM systems such as SVN, CVS, Git, and so on.
- This supports common test frameworks such as Junit, Selenium, and so on.

Jenkins speeds up the application development process through automation across different phases such as build, test, code analysis, and so on. It also enables users to achieve end-to-end automation for an application delivery life cycle.

What this book covers

Chapter 1, Exploring Jenkins, describes in detail the basics of continuous integration and provides an overview of Jenkins. This chapter also describes installation and configuration of Jenkins. It takes a jump-start tour through some of the key features of Jenkins and plugin installations as well. It will also cover the deployment pipeline and the rest of the chapters will cover implementing it.

Chapter 2, Installation and Configuration of Code Repository and Build Tools, describes in detail on how to prepare runtime environment for application life cycle management and configure it with Jenkins – an open source continuous integration tool. It will cover how to integrate Eclipse and code repository such as SVN and Git to create a base for continuous integration in the deployment pipeline, which is explained in *Chapter 1, Exploring Jenkins*.

Chapter 3, Integration of Jenkins, SVN, and Build Tools, describes in detail on how to create and configure build jobs for Java applications, and how to run build jobs and unit test cases. It covers all aspects of running a build to create a distribution file or WAR file for deployment.

Chapter 4, Implementing Automated Deployment, covers one step forward in the deployment pipeline by deploying artifacts in the local or remote application server. It will give insight into automated deployment and continuous delivery process, and also cover how to deploy applications on a public cloud platform using Jenkins.

Chapter 5, Hosted Jenkins, describes how to use Jenkins on Platform as a Service (PaaS) model, which is provided by popular PaaS providers such as Red Hat OpenShift and CloudBees. Considering CloudBees, it also covers details on how various customers are using Jenkins based on their requirements. This chapter will explore details on how to use Cloud-related plugins in Jenkins for an effective use of Jenkins.

Chapter 6, Managing Code Quality and Notifications, covers how to integrate static code analysis behavior into Jenkins. Code quality is an extremely vital feature that impacts an application's effectiveness, and by integrating it with Sonar, CheckStyle, FindBug, and other tools, you can get an insight into problematic portions of code.

Chapter 7, Managing and Monitoring Jenkins, gives an insight into management of Jenkins nodes and monitoring them with Java Melody to provide details on utilization of resources. It also covers how to monitor build jobs configured for Java applications and managing those configurations by keeping its backup. This chapter discusses the basic security configuration that is available in Jenkins for better access control and authorization.

Chapter 8, Beyond Basics of Jenkins – Leveraging "Must-have" Plugins, covers the advanced usage of Jenkins that are extremely useful in specific scenarios. Scenario-based use cases and usage of specific plugins that help development and operations teams are covered here for better utilization of Jenkins.

What you need for this book

This book assumes that you are familiar with at least Java programming language. Knowledge of core Java and JEE is essential. Having a strong understanding of program logic will provide you with the background to be productive with Jenkins while using plugins of writing commands for shell.

As an application development life cycle will cover lots of tools in general, it is essential to have some knowledge of repositories such as SVN, Git, and so on; IDE tools such as Eclipse; and build tools such as Ant and Maven.

Knowledge of code analysis tools will make jobs easier in configuration and integration; however, it is not extremely vital to perform the exercises given in the book. Most of the configuration steps are mentioned clearly.

You will be walked through the steps required to install Jenkins on a Windows- and Linux-based host. In order to be immediately successful, you will need administrative access to a host that runs a modern version of Linux; CentOS 6.x is what will be used for demonstration purposes. If you are a more experienced reader, then a recent release of almost any distribution will work just as well (but you may be required to do a little bit of extra work that is not outlined in this book). If you do not have access to a dedicated Linux host, a virtual host (or hosts) running inside of virtualization software such as VirtualBox or VMware workstation will work.

Additionally, you will need access to the Internet to download plugins that you do not already have and also have Jenkins installed.

Who this book is for

This book targets developers and system administrators who are involved in the application development life cycle and are looking to automate it. Developers, technical leads, testers, and operational professionals are the target readers to jump-start Jenkins. Readers are aware of the issues faced by the development and operations team as they are stakeholders in the application life cycle management process. The reasons to jump-start Jenkins are to understand the importance of contribution in continuous integration, automated test case execution, and continuous delivery for an effective application life cycle management.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:

"Commit by executing `git commit -m "Initial Commit" -a.`"

Any command-line input or output is written as follows:

```
[root@localhost testmit]# service httpd restart
```

```
Stopping httpd:
```

```
[ OK ]
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Once the build has succeeded, verify **Workspace** in the build job."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Exploring Jenkins

"Continuous effort – not strength or intelligence – is the key to unlocking our potential."

– Winston Churchill

Jenkins is an open source application written in Java. It is one of the most popular **continuous integration (CI)** tools used to build and test different kinds of projects. In this chapter, we will have a quick overview of Jenkins, essential features, and its impact on DevOps culture. Before we can start using Jenkins, we need to install it. In this chapter, we have provided a step-by-step guide to install Jenkins. Installing Jenkins is a very easy task and is different from the OS flavors.

We will also learn the basic configuration of Jenkins. We will take a quick tour of some key sections of the Jenkins UI and plugin installations as well. This chapter will also cover the DevOps pipeline and how the rest of the chapters will cover implementing it.

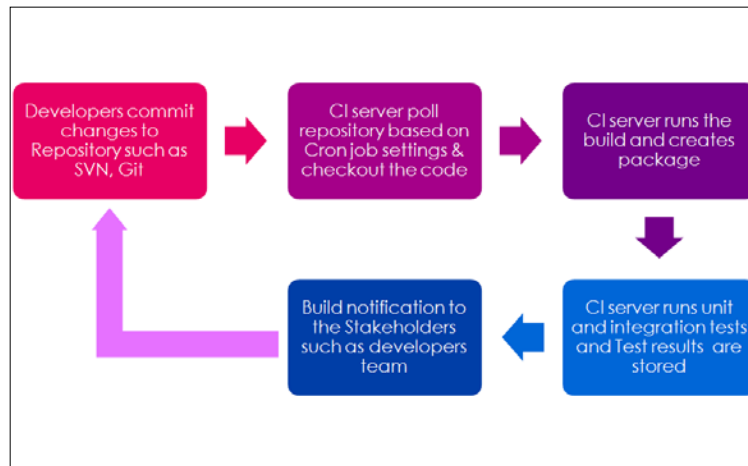
To be precise, we will discuss the following topics in this chapter:

- Introduction to Jenkins and its features
- Installation of Jenkins on Windows and the CentOS operating system
- A jump-start tour of the Jenkins dashboard
- How to change configuration settings in Jenkins
- What is the deployment pipeline

On your mark, get set, go!

Introduction to Jenkins and its features

Let's first understand what continuous integration is. CI is one of the most popular application development practices in recent times. Developers integrate bug fix, new feature development, or innovative functionality in code repository. The CI tool verifies the integration process with an automated build and automated test execution to detect issues with the current source of an application, and provide quick feedback.



Jenkins is a simple, extensible, and user-friendly open source tool that provides CI services for application development. Jenkins supports SCM tools such as StarTeam, Subversion, CVS, Git, AccuRev and so on. Jenkins can build Freestyle, Apache Ant, and Apache Maven-based projects.

The concept of plugins makes Jenkins more attractive, easy to learn, and easy to use. There are various categories of plugins available such as Source code management, Slave launchers and controllers, Build triggers, Build tools, Build notifies, Build reports, other post-build actions, External site/tool integrations, UI plugins, Authentication and user management, Android development, iOS development, .NET development, Ruby development, Library plugins, and so on.

Jenkins defines interfaces or abstract classes that model a facet of a build system. Interfaces or abstract classes define an agreement on what needs to be implemented; Jenkins uses plugins to extend those implementations.



To learn more about all plugins, visit <https://wiki.jenkins-ci.org/x/GIAL>.

To learn how to create a new plugin, visit <https://wiki.jenkins-ci.org/x/TYAL>.

To download different versions of plugins, visit <https://updates.jenkins-ci.org/download/plugins/>.

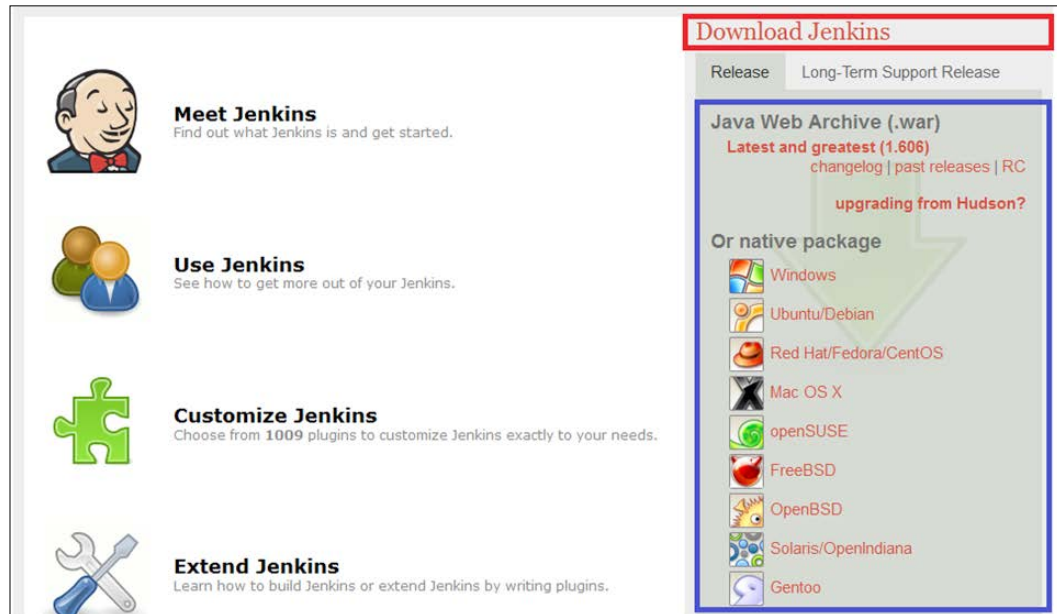
Features

Jenkins is one of the most popular CI servers in the market. The reasons for its popularity are as follows:

- Easy installation on different operating systems.
- Easy upgrades—Jenkins has very speedy release cycles.
- Simple and easy-to-use user interface.
- Easily extensible with the use of third-party plugins—over 400 plugins.
- Easy to configure the setup environment in the user interface. It is also possible to customize the user interface based on likings.
- The master slave architecture supports distributed builds to reduce loads on the CI server.
- Jenkins is available with test harness built around JUnit; test results are available in graphical and tabular forms.
- Build scheduling based on the cron expression (to know more about cron, visit <http://en.wikipedia.org/wiki/Cron>).
- Shell and Windows command execution in prebuild steps.
- Notification support related to the build status.

Installation of Jenkins on Windows and CentOS

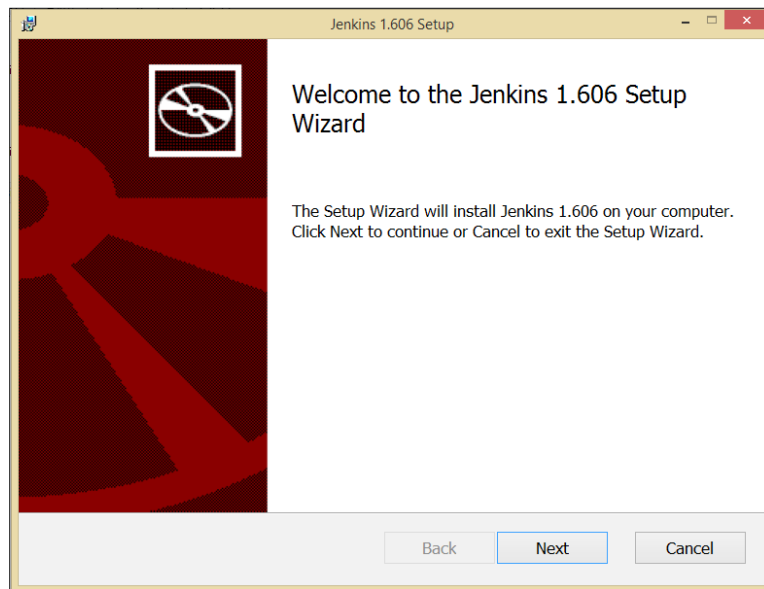
1. Go to <https://jenkins-ci.org/>. Find the **Download Jenkins** section on the home page of Jenkins's website.



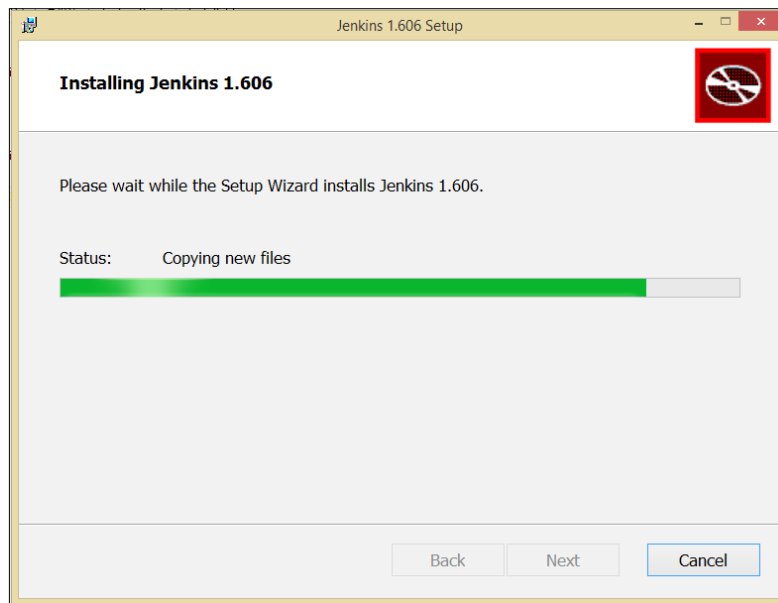
2. Download the war file or native packages based on your operating system. A Java installation is needed to run Jenkins.
3. Install Java based on your operating system and set the JAVA_HOME environment variable accordingly.

Installing Jenkins on Windows

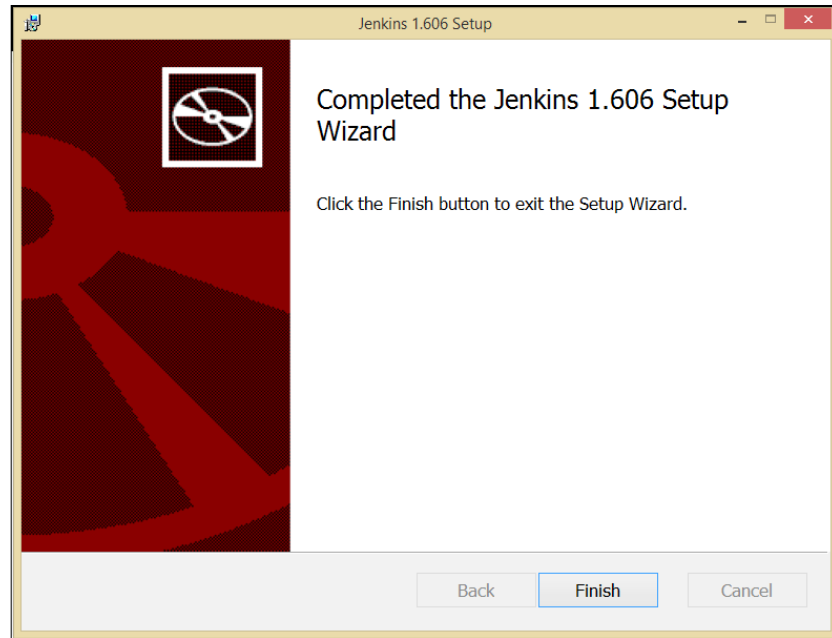
1. Select the native package available for Windows. It will download jenkins-1.xxx.zip. In our case, it will download jenkins-1.606.zip. Extract it and you will get setup.exe and jenkins-1.606.msi files.
2. Click on setup.exe and perform the following steps in sequence. On the welcome screen, click **Next**:



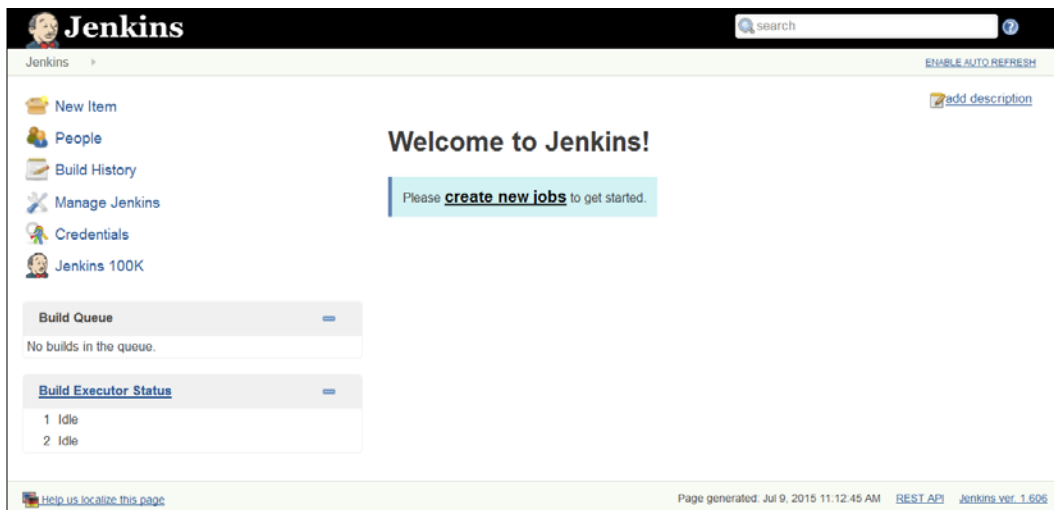
3. Select the destination folder and click on **Next**.
4. Click on **Install** to begin installation. Please wait while the Setup Wizard installs Jenkins.



5. Once the Jenkins installation is completed, click on the **Finish** button.

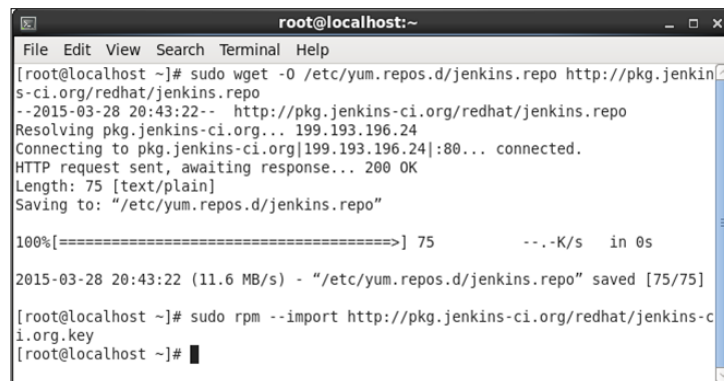


6. Verify the Jenkins installation on the Windows machine by opening URL `http://<ip_address>:8080` on the system where you have installed Jenkins.



Installation of Jenkins on CentOS

1. To install Jenkins on CentOS, download the Jenkins repository definition to your local system at `/etc/yum.repos.d/` and import the key.
2. Use the `wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo` command to download repo.



```

root@localhost:~
File Edit View Search Terminal Help
[root@localhost ~]# sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
--2015-03-28 20:43:22-- http://pkg.jenkins-ci.org/redhat/jenkins.repo
Resolving pkg.jenkins-ci.org... 199.193.196.24
Connecting to pkg.jenkins-ci.org[199.193.196.24]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 75 [text/plain]
Saving to: "/etc/yum.repos.d/jenkins.repo"

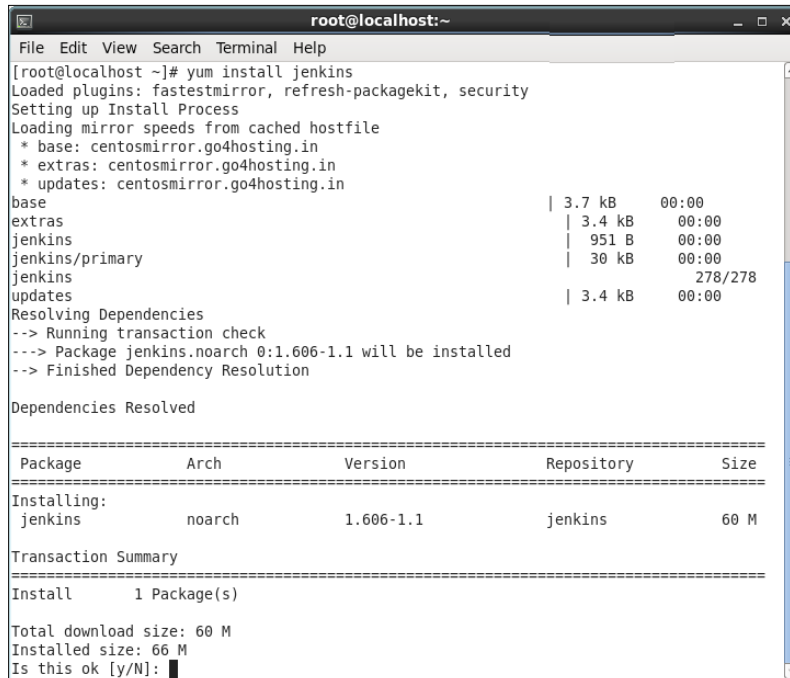
100%[=====] 75      --.-K/s  in 0s

2015-03-28 20:43:22 (11.6 MB/s) - "/etc/yum.repos.d/jenkins.repo" saved [75/75]

[root@localhost ~]# sudo rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
[root@localhost ~]#

```

3. Now, run `yum install Jenkins`; it will resolve dependencies and prompt for installation.



```

root@localhost:~
File Edit View Search Terminal Help
[root@localhost ~]# yum install jenkins
Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Loading mirror speeds from cached hostfile
* base: centosmirror.go4hosting.in
* extras: centosmirror.go4hosting.in
* updates: centosmirror.go4hosting.in
base                                     | 3.7 kB    00:00
extras                                 | 3.4 kB    00:00
jenkins                                | 951 B     00:00
jenkins/primary                        | 30 kB     00:00
jenkins                                |          278/278
updates                                | 3.4 kB    00:00
Resolving Dependencies
--> Running transaction check
--> Package jenkins.noarch 0:1.606-1.1 will be installed
--> Finished Dependency Resolution

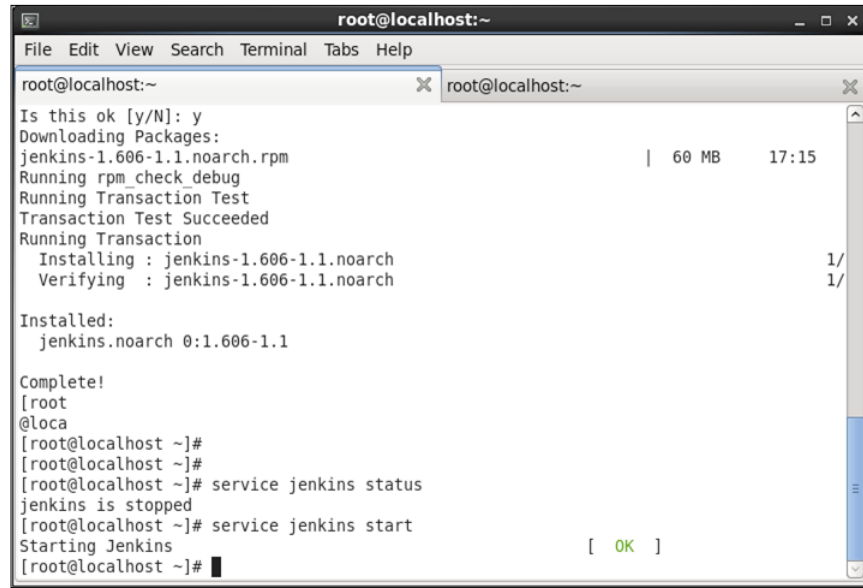
Dependencies Resolved

=====
Package             Arch      Version      Repository      Size
=====
Installing:
jenkins             noarch    1.606-1.1    jenkins         60 M
=====

Transaction Summary
=====
Install      1 Package(s)
=====
Total download size: 60 M
Installed size: 66 M
Is this ok [y/N]:

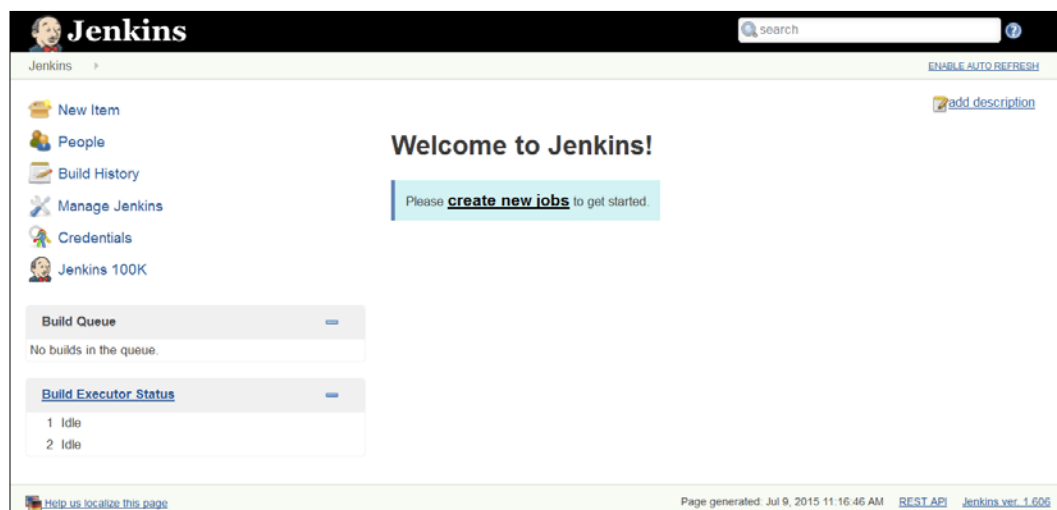
```

4. Reply with `y` and it will download the required package to install Jenkins on CentOS. Verify the Jenkins status by issuing the `service jenkins status` command. Initially, it will be stopped. Start Jenkins by executing `service jenkins start` in the terminal.



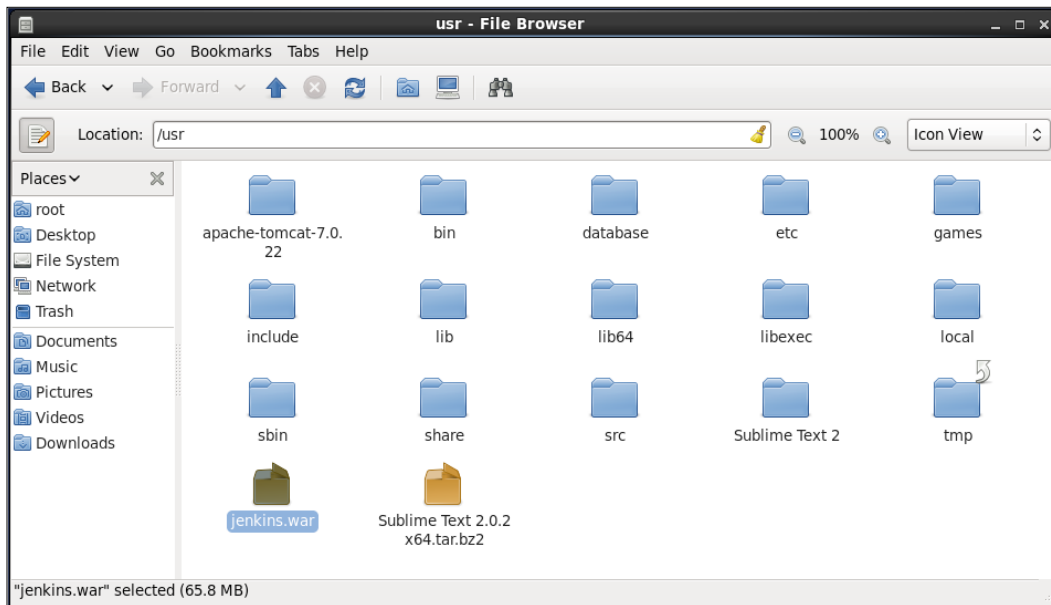
```
root@localhost:~  
File Edit View Search Terminal Tabs Help  
root@localhost:~  
Is this ok [y/N]: y  
Downloading Packages:  
jenkins-1.606-1.1.noarch.rpm | 60 MB 17:15  
Running rpm_check debug  
Running Transaction Test  
Transaction Test Succeeded  
Running Transaction  
Installing : jenkins-1.606-1.1.noarch 1/  
Verifying : jenkins-1.606-1.1.noarch 1/  
  
Installed:  
jenkins.noarch 0:1.606-1.1  
  
Complete!  
[root  
@loca  
[root@localhost ~]#  
[root@localhost ~]#  
[root@localhost ~]# service jenkins status  
jenkins is stopped  
[root@localhost ~]# service jenkins start  
Starting Jenkins [ OK ]  
[root@localhost ~]#
```

5. Verify the Jenkins installation on the CentOS machine by opening the URL `http://<ip_address>:8080` on the system where you have installed Jenkins.

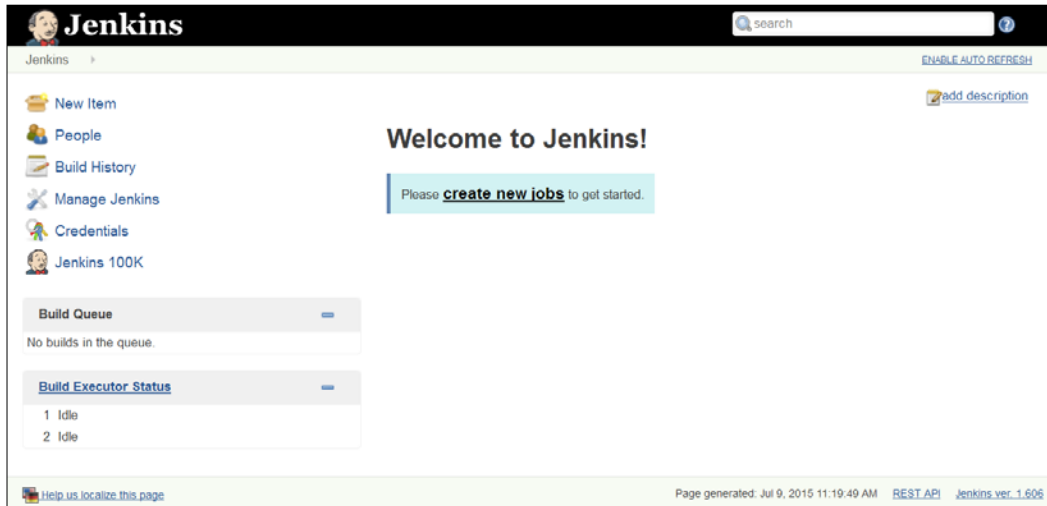


Installation of Jenkins as a web application

1. Download **Java Web Archive (.war)** (latest and greatest (1.606)) from <http://jenkins-ci.org/>.
2. Copy `jenkins.war` into your virtual or physical machine. Open Command Prompt or a terminal based on the operation system. In our case, we will copy it into a directory of a CentOS virtual machine.

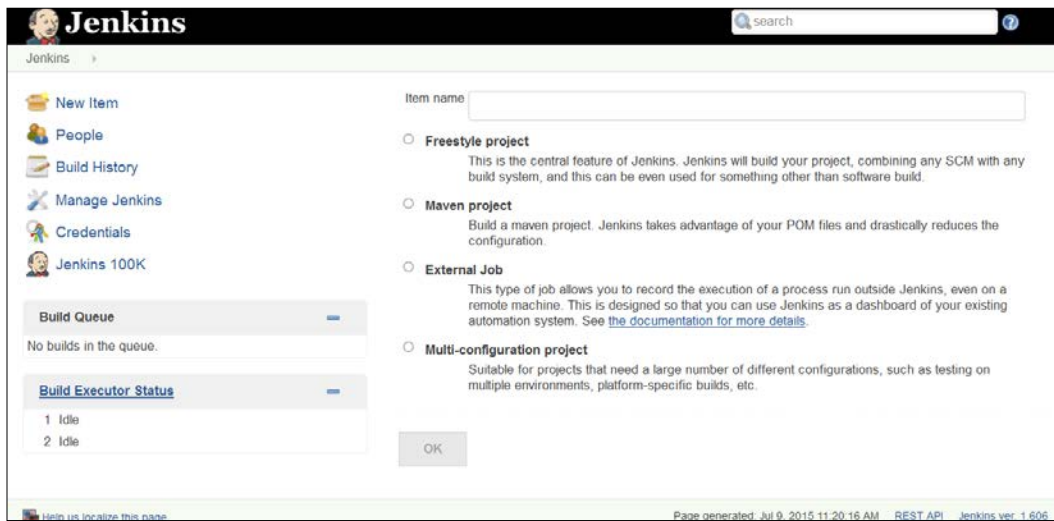


3. Open Command Prompt and execute the `java -jar Jenkins.war` command. Verify the Jenkins installation on the system by opening the `http://<ip_address>:8080` URL on the system where you have installed Jenkins.



A jump-start tour of the Jenkins dashboard

1. On the Jenkins dashboard, click on **Create new jobs** or on **New Item** to create Freestyle- or Maven-based projects for CI.



The image shows the Jenkins 'New Item' dialog. On the left is a sidebar with navigation links: New Item, People, Build History, Manage Jenkins, Credentials, and Jenkins 100K. Below these are sections for 'Build Queue' (showing 'No builds in the queue') and 'Build Executor Status' (showing two idle executors). The main area has a search bar and a list of project types: Freestyle project, Maven project, External Job, and Multi-configuration project, each with a brief description. An 'Item name' input field is at the top right, and an 'OK' button is at the bottom right. The footer shows the page generation date and Jenkins version.

Jenkins

Item name

☐ **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☐ **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

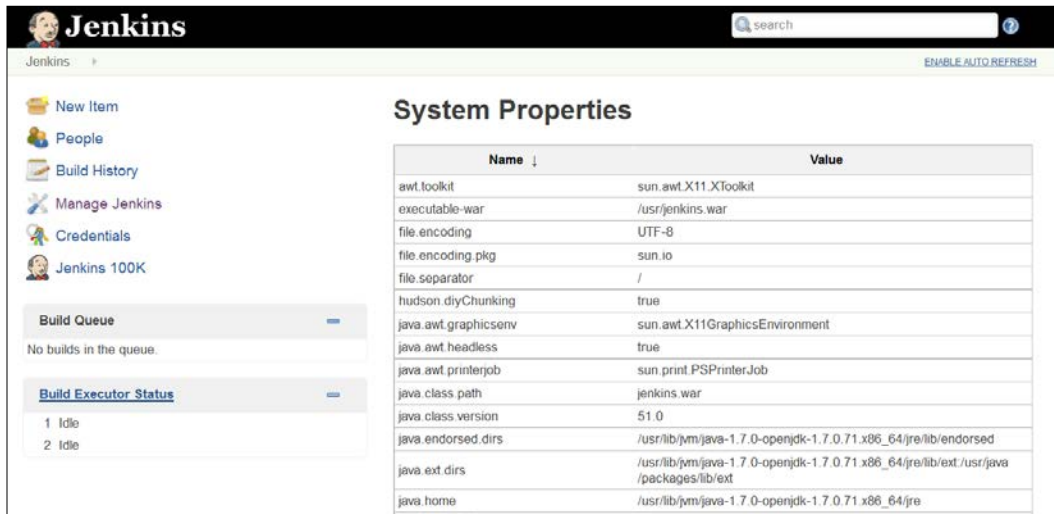
☐ **External Job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

☐ **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

OK

Page generated: Jul 9, 2015 11:20:16 AM REST API Jenkins ver 1.606

- To verify system properties, visit `http://<ip_address>:8080/systeminfo` or click on **Manage Jenkins**, and then click on **System Information** to get environmental information to assist troubleshooting.



The image shows the Jenkins 'System Properties' page. It features the same sidebar as the previous screenshot. The main content area is titled 'System Properties' and contains a table with system information. A 'ENABLE AUTO REFRESH' link is in the top right corner.

Jenkins

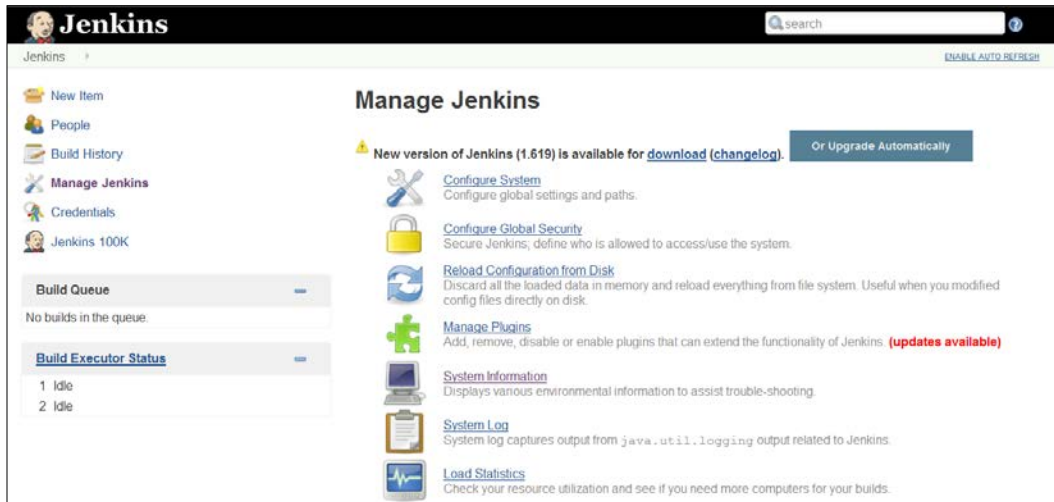
ENABLE AUTO REFRESH

System Properties

Name	Value
awt.toolkit	sun.awt.X11.XToolkit
executable-war	/usr/jenkins.war
file.encoding	UTF-8
file.encoding.pkg	sun.io
file.separator	/
hudson.diyChunking	true
java.awt.graphicsenv	sun.awt.X11GraphicsEnvironment
java.awt.headless	true
java.awt.printerjob	sun.print.PSPrinterJob
java.class.path	jenkins.war
java.class.version	51.0
java.endorsed.dirs	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.71.x86_64/jre/lib/endorsed
java.ext.dirs	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.71.x86_64/jre/lib/ext:/usr/java/packages/lib/ext
java.home	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.71.x86_64/jre

How to change configuration settings in Jenkins

1. Click on the **Manage Jenkins** link on the dashboard to configure system, security, to manage plugins, slave nodes, credentials, and so on.



2. Click on the **Configure System** link to configure Java, Ant, Maven, and other third-party products' related information.

Jenkins > configuration

JDK

JDK installations Add JDK

List of JDK installations on this system

Ant

Ant installations Add Ant

List of Ant installations on this system

Maven

Maven installations Add Maven

List of Maven installations on this system

Maven Project Configuration

Global MAVEN_OPTS ?

Local Maven Repository Default (~/.m2/repository) ?

☒ Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project. ?

Jenkins Location

Save Apply

- Jenkins uses Groovy as its scripting language. To execute the arbitrary script for administration/trouble-shooting/diagnostics on the Jenkins dashboard, go to the **Manage Jenkins** link on the dashboard, click on **Script Console**, and run `println(Jenkins.instance.pluginManager.plugins)`.

Jenkins

Search

New Item
People
Build History
Manage Jenkins
Credentials
Jenkins 100K

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

Script Console

Type in an arbitrary [Groovy script](#) and execute it on the server. Useful for trouble-shooting and diagnostics. Use the 'println' command to see the output (if you use System.out, it will go to the server's stdout, which is harder to see.) Example:

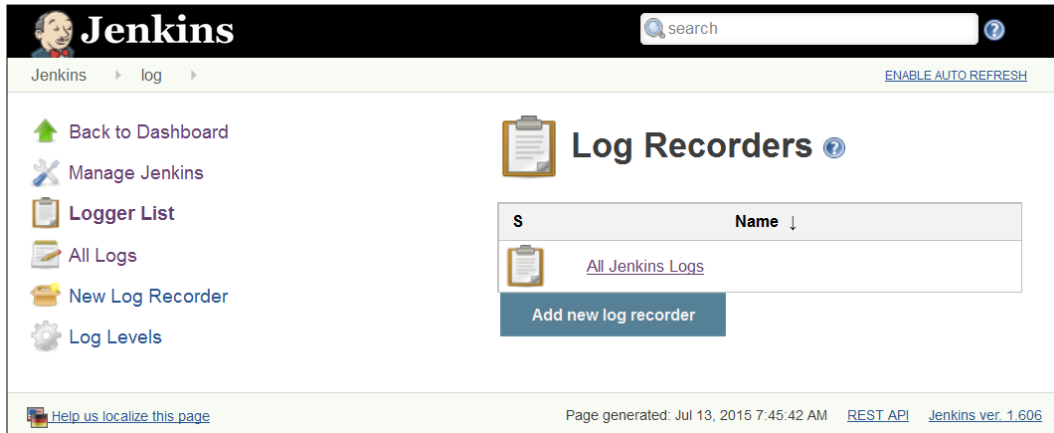
```
println(Jenkins.instance.pluginManager.plugins)
```

All the classes from all the plugins are visible. `jenkins.*`, `jenkins.model.*`, `hudson.*`, and `hudson.model.*` are pre-imported.

```
1 println(Jenkins.instance.pluginManager.plugins)
```

Run

- To verify the system log, go to the **Manage Jenkins** link on the dashboard and click on the **System Log** link or visit <http://localhost:8080/log/all>.



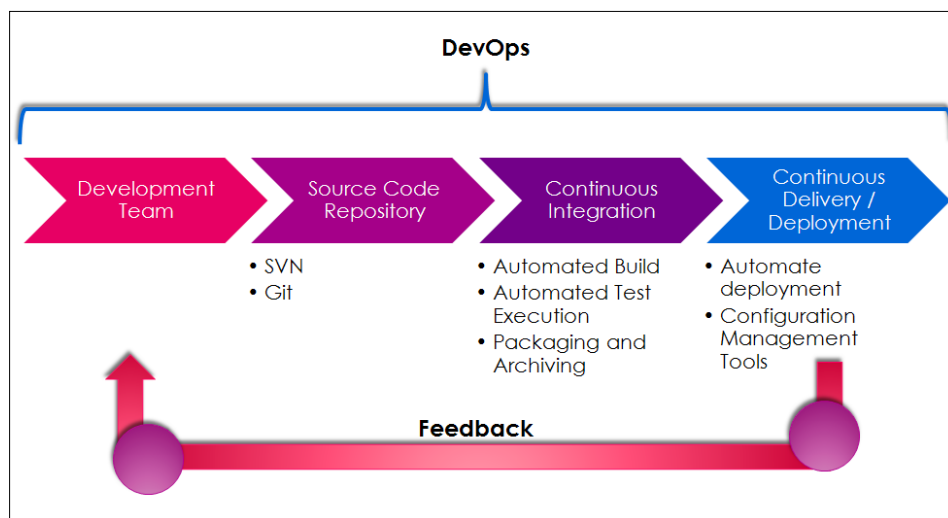
- To get more information on third-party libraries – version and license information in Jenkins, go to the **Manage Jenkins** link on the dashboard and click on the **About Jenkins** link.

The screenshot shows the 'About Jenkins 1.606' page. It includes a description of Jenkins as a community-developed open-source continuous integration server and a list of third-party libraries it depends on. The table below lists these libraries with their names, Maven IDs, and licenses.

Name	Maven ID	License
Jenkins war	org.jenkins-ci.main:jenkins-war:1.606	The MIT license
Spring Framework: Web MVC	org.springframework:spring-webmvc:2.5.6.SEC03	The Apache Software License, Version 2.0
Commons BeanUtils	commons-beanutils:commons-beanutils:1.8.3	The Apache Software License, Version 2.0
Bouncy Castle Provider	org.bouncycastle:bcprov-jdk15on:1.47	Bouncy Castle Licence
oro	oro:oro:2.0.8	The Apache Software License, Version 2.0
Task Reactor	org.jenkins-ci:task-reactor:1.4	MIT License
dom4j	org.jenkins-ci.dom4j:dom4j:1.6.1-jenkins-4	BSD License
Groovy	org.codehaus.groovy:groovy-all:1.8.9	The Apache Software License, Version 2.0

What is the deployment pipeline?

The application development life cycle is a traditionally lengthy and a manual process. In addition, it requires effective collaboration between development and operations teams. The deployment pipeline is a demonstration of automation involved in the application development life cycle containing the automated build execution and test execution, notification to the stakeholder, and deployment in different runtime environments. Effectively, the deployment pipeline is a combination of CI and continuous delivery, and hence is a part of DevOps practices. The following diagram depicts the deployment pipeline process:



Members of the development team check code into a source code repository. CI products such as Jenkins are configured to poll changes from the code repository. Changes in the repository are downloaded to the local workspace and Jenkins triggers an automated build process, which is assisted by Ant or Maven. Automated test execution or unit testing, static code analysis, reporting, and notification of successful or failed build process are also part of the CI process.

Once the build is successful, it can be deployed to different runtime environments such as testing, preproduction, production, and so on. Deploying a war file in terms of the JEE application is normally the final stage in the deployment pipeline.

One of the biggest benefits of the deployment pipeline is the faster feedback cycle. Identification of issues in the application at early stages and no dependencies on manual efforts make this entire end-to-end process more effective.

In the next chapters, we will see how Jenkins can be used for implementing CI practices in modernizing IT.



To read more, visit <http://martinfowler.com/bliki/DeploymentPipeline.html> and <http://www.informit.com/articles/article.aspx?p=1621865&seqNum=2>.

Self-test questions

Q1. What is Jenkins?

1. A continuous integration product
2. A continuous delivery product

Q2. What makes Jenkins extensible?

1. Plugins
2. Open Source Distribution

Q3. Which command is used to run the Jenkins installation file in the `war` format?

1. `java -jar Jenkins.war`
2. `java -j Jenkins.war`

Q4. How do we get system information on the Jenkins dashboard?

1. Visit `http://<ip_address>:8080/manage`
2. Visit `http://<ip_address>:8080/systeminfo`

Q5. How do we change global settings for configuration on the Jenkins dashboard?

1. Click on the **Manage Jenkins** link on the dashboard
2. Click on the **Credentials** link on the dashboard

Q6. What is the deployment pipeline?

1. Continuous Integration Practices
2. Continuous Delivery Practices
3. Demonstration of automation involved in the application development life cycle
4. None of the above

Q7. Explain the benefits of the deployment pipeline?

1. Faster feedback cycle
2. Identification of issues in an application at early stages
3. No dependencies on manual efforts
4. All of the above

Summary

Congratulations! We reached the end of this chapter and hence we have Jenkins installed on our physical or virtual machine, and you are ready to go to the next chapter. Till now, we covered the basics of CI and the introduction to Jenkins and its features. We completed the installation of Jenkins on Windows and CentOS platforms. We also completed a quick tour of features available in Jenkins's dashboard. In addition to this, we discussed the deployment pipeline and its importance in CI.

Now that we are able to use our CI server, Jenkins, we can begin creating a job and verify how Jenkins works.

2

Installation and Configuration of Code Repository and Build Tools

"Life is really simple, but we insist on making it complicated"

– Confucius

We looked at the deployment pipeline in the last chapter in which the source code repository and automated build form a significant part. SVN, Git, CVS, and StarTeam are some of the popular code repositories that manage changes to code, artifacts, or documents, while Ant and Maven are popular build automation tools for Java applications.

This chapter describes in detail how to prepare a runtime environment for life cycle management with a Java application and configure it with Jenkins. It will cover how to integrate Eclipse and code repositories such as SVN to create a base for continuous integration. The following is the list of topics covered in this chapter:

- Overview of a build in Jenkins and its requirements
- Installing Java and configuring environment variables
- SVN installation, configuration, and operations on CentOS and Windows
- Installing Ant
- Configuring Ant, Maven, and JDK in Jenkins
- Integrating Eclipse with code repositories
- Installing and configuring Git
- Creating a new build job in Jenkins with Git

An overview of a build in Jenkins and its requirements

To explain continuous integration, we are going to use a code repository installed on a physical machine or laptop while Jenkins is installed on a virtual machine, as suggested in different ways in *Chapter 1, Exploring Jenkins*. The following figure depicts the setup of the runtime environment:



We saw in *Chapter 1, Exploring Jenkins*, that the **Manage Jenkins** link on the dashboard is used to configure the system. Click on the **Configure System** link to configure Java, Ant, Maven, and other third-party product-related information. We can create a virtual machine with Virtual box or the VMware workstation. We need to install all required software to provide a runtime environment for continuous integration. We assume that Java is already installed in the system.

Installing Java and configuring environment variables

If Java is not already installed in the system then you can install it as follows:

Find the Java related packages available in CentOS repository and locate the appropriate package to install.

```
[root@localhost ~]# yum search java
```

```
Loaded plugins: fastestmirror, refresh-packagekit, security
```

```

.
.
ant-javamail.x86_64 : Optional javamail tasks for ant
eclipse-mylyn-java.x86_64 : Mylyn Bridge: Java Development
.
.
java-1.5.0-gcj.x86_64 : JPackage runtime compatibility layer for GCJ
java-1.5.0-gcj-devel.x86_64 : JPackage development compatibility layer
for GCJ
java-1.5.0-gcj-javadoc.x86_64 : API documentation for libgcj
java-1.6.0-openjdk.x86_64 : OpenJDK Runtime Environment
java-1.6.0-openjdk-devel.x86_64 : OpenJDK Development Environment
java-1.6.0-openjdk-javadoc.x86_64 : OpenJDK API Documentation
java-1.7.0-openjdk.x86_64 : OpenJDK Runtime Environment
jcommon-serializer.x86_64 : JFree Java General Serialization Framework
.
.

```

```

Install the identified package java-1.7.0-openjdk.x86_64
[root@localhost ~]# yum install java-1.7.0-openjdk.x86_64
Loaded plugins: fastestmirror, refresh-packagekit, security
No such command: in. Please use /usr/bin/yum -help

```

Now install Java package available in the local repositories by executing yum install command as follows:

```

[root@localhost ~]# yum install java-1.7.0-openjdk.x86_64
Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package java-1.7.0-openjdk.x86_64 1:1.7.0.3-2.1.el6.7 will be
installed
--> Finished Dependency Resolution

```

Dependencies Resolved

```

.
.

```

```
Install          1 Package(s)

Total download size: 25 M
Installed size: 89 M
Is this ok [y/N]: y
Downloading Packages:
java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86_64.rpm
| 25 MB      00:00
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction

  Installing : 1:java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86_64
1/1
  Verifying   : 1:java-1.7.0-openjdk-1.7.0.3-2.1.el6.7.x86_64
1/1

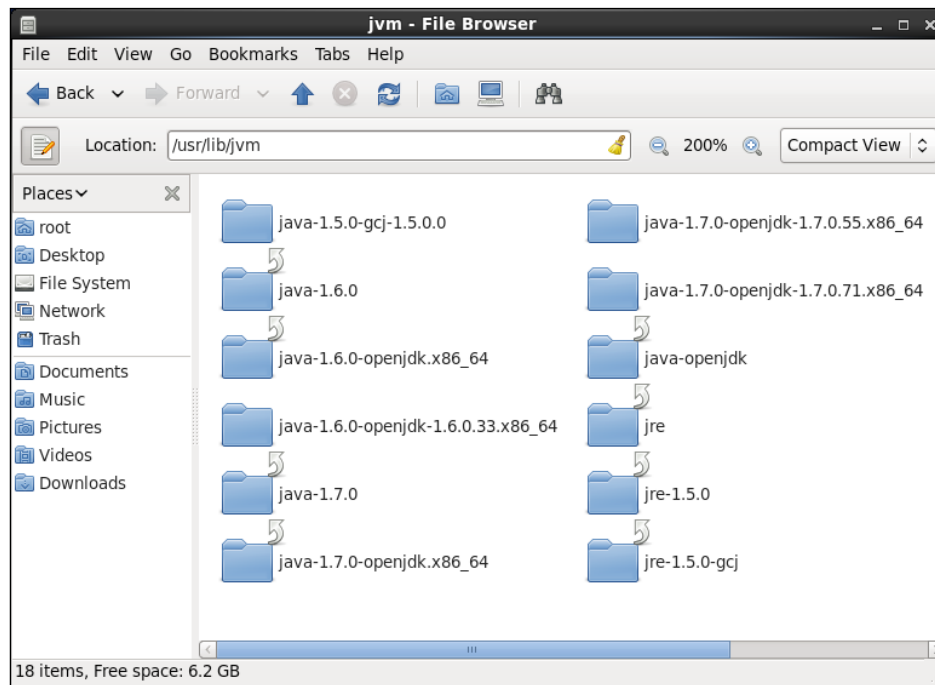
Installed:
  java-1.7.0-openjdk.x86_64 1:1.7.0.3-2.1.el6.7
Complete!
```

Java is installed successfully from the local repository.

Configure environment variables

The following are the steps to configure the environment variables:

1. Set `JAVA_HOME` and `JRE_HOME` variables
2. Go to `/root`
3. Press `Ctrl + H` to list hidden files
4. Find `.bash_profile` and edit it by appending the Java path, as shown in the following screenshot:



Installing, configuring, and operating SVN on CentOS and Windows

Install SVN from the local repository on CentOS.

Installing SVN on CentOS

To install SVN on a CentOS machine, execute the `yum install mod_dav_svn subversion` command as follows:

```
[root@localhost ~]# yum install mod_dav_svn subversion
Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mod_dav_svn.x86_64 0:1.6.11-7.el6 will be installed
```

```
--> Package subversion.x86_64 0:1.6.11-7.el6 will be installed
--> Processing Dependency: perl.URI >= 1.17 for package:
subversion-1.6.11-7.el6.x86_64
--> Running transaction check
--> Package perl.URI.noarch 0:1.40-2.el6 will be installed
--> Finished Dependency Resolution
```

Dependencies Resolved

.

Installed:

```
mod_dav_svn.x86_64 0:1.6.11-7.el6
subversion.x86_64 0:1.6.11-7.el6
```

Dependency Installed:

```
perl.URI.noarch 0:1.40-2.el6
```

Complete!

```
[root@localhost ~]#
```

Configuring SVN

Create the password file using the `htpasswd` command. Initially use the `-cm` arguments. This creates the file and also encrypts the password with MD5. If you need to add users, make sure you simply use the `-m` flag, and not the `-c`, after the initial creation.

```
[root@localhost conf.d]# htpasswd -cm /etc/svn-auth-conf yourusername
New password:
Re-type new password:
Adding password for user yourusername
[root@localhost conf.d]#
```

```
[root@localhost conf.d]# htpasswd -cm /etc/svn-auth-conf mitesh
New password:
Re-type new password:
Adding password for user mitesh
[root@localhost conf.d]#
```

Now configure SVN in Apache to integrate both. Edit `/etc/httpd/conf.d/subversion.conf`. The location is what Apache will pass in the URL bar.

```
LoadModule dav_svn_module      modules/mod_dav_svn.so
LoadModule authz_svn_module    modules/mod_authz_svn.so

#
# Example configuration to enable HTTP access for a directory
# containing Subversion repositories, "/var/www/svn".  Each repository
# must be both:
#
#   a) readable and writable by the 'apache' user, and
#
#   b) labelled with the 'httpd_sys_content_t' context if using
#   SELinux
#

#
# To create a new repository "http://localhost/repos/stuff" using
# this configuration, run as root:
#
#   # cd /var/www/svn
#   # svnadmin create stuff
#   # chown -R apache.apache stuff
#   # chcon -R -t httpd_sys_content_t stuff
#

<Location />
    DAV svn
    SVNParentPath /var/www/svn/
#
# Limit write permission to list of valid users.
# <LimitExcept GET PROPFIND OPTIONS REPORT>
#     # Require SSL connection for password protection.
#     # SSLRequireSSL
#
```

```
AuthType Basic
SVNListParentPath on
AuthName "Subversion repos"
AuthUserFile /etc/svn-auth-conf
Require valid-user
# </LimitExcept>
</Location>
```

Now all configurations are completed. Let's perform operations on SVN.

SVN operations

Create the actual repository to perform SVN operations on the CentOS virtual machine.

```
[root@localhost ~] cd /var/www/ -- Or wherever you placed your path above
[root@localhost ~] mkdir svn
[root@localhost ~] cd svn
[root@localhost ~] svnadmin create repos
[root@localhost ~] chown -R apache:apache repos
[root@localhost ~] service httpd restart
```

Import a directory into SVN

Create a sample folder structure to test SVN operations. Create the mytestproj directory with sub-directories named main, configurations, and resources. Create sample files in each sub-directory.

```
[root@localhost mytestproj]# svn import /tmp/mytestproj/ file:///var/www/
svn/repos/mytestproj -m "Initial repository layout for mytestproj"
Adding          /tmp/mytestproj/main
Adding          /tmp/mytestproj/main/mainfile1.cfg
Adding          /tmp/mytestproj/configurations
Adding          /tmp/mytestproj/configurations/testconfl.cfg
Adding          /tmp/mytestproj/resources
Adding          /tmp/mytestproj/resources/testresources1.cfg
Committed revision 1.
```

Verify the repository from a web browser: <http://localhost/repos>.

Check out from SVN

To check out source code from the repository, perform the following operations:

1. Start httpd service.

```
[root@localhost testmit]# service httpd restart
Stopping httpd:
[ OK ]

Starting httpd: httpd: Could not reliably determine the server's
fully qualified domain name, using localhost.localdomain for
ServerName
[ OK ]
```

2. Check out the source code.

```
[root@localhost testmit]# svn co http://localhost/repos/mytestproj
Authentication realm: <http://localhost:80> Subversion repos
Password for 'root':
Authentication realm: <http://localhost:80> Subversion repos
Username: mitesh
Password for 'mitesh':xxxxxxxxx
```

```
-----
-----
ATTENTION! Your password for authentication realm:
```

```
<http://localhost:80> Subversion repos
```

```
can only be stored to disk unencrypted! You are advised to
configure your system so that Subversion can store passwords
encrypted, if possible. See the documentation for details.
```

3. You can avoid future appearances of this warning by setting the value of the store-plaintext-passwords option to either yes or no in /root/.subversion/servers.

```
-----
-----
Store password unencrypted (yes/no)? no
```

```
A mytestproj/main
```

```
A mytestproj/main/mainfile1.cfg
```

```
A mytestproj/configurations
```

A `mytestproj/configurations/testconf1.cfg`

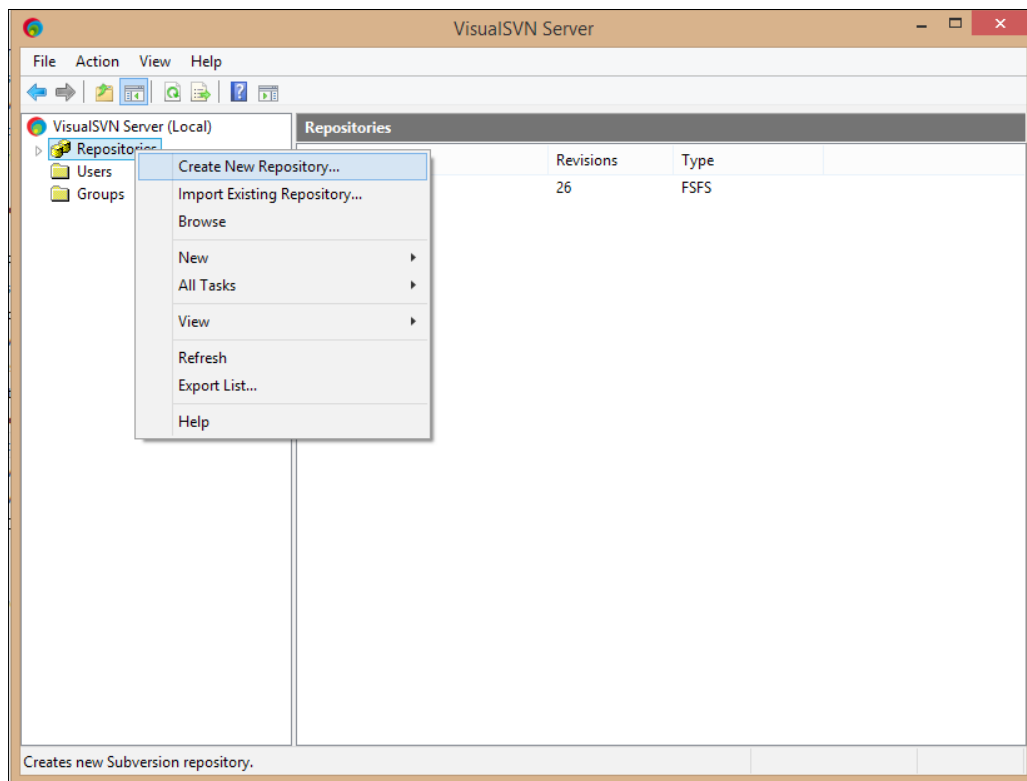
A `mytestproj/options`

A `mytestproj/options/testopts1.cfg`

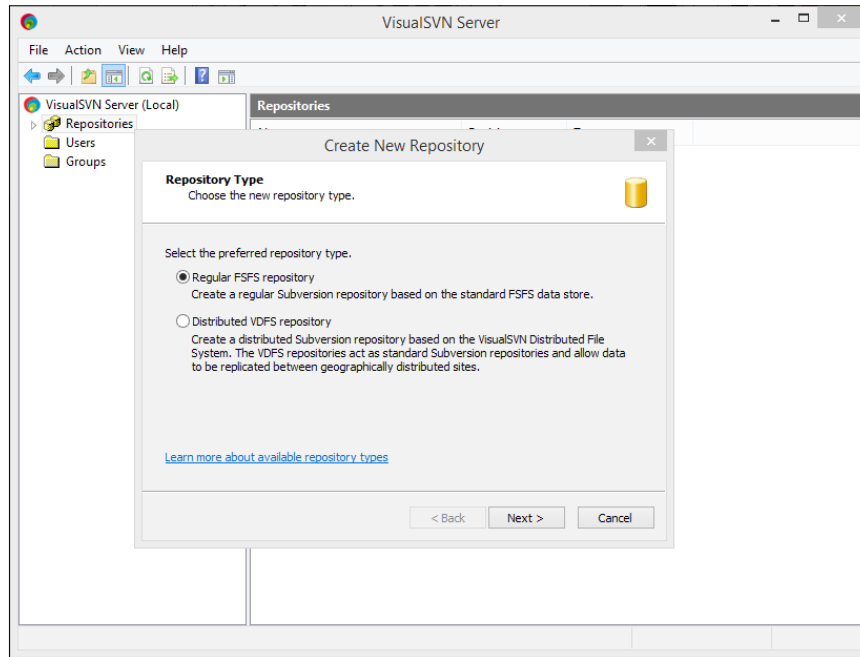
Checked out revision 1.

VisualSVN Server on Windows

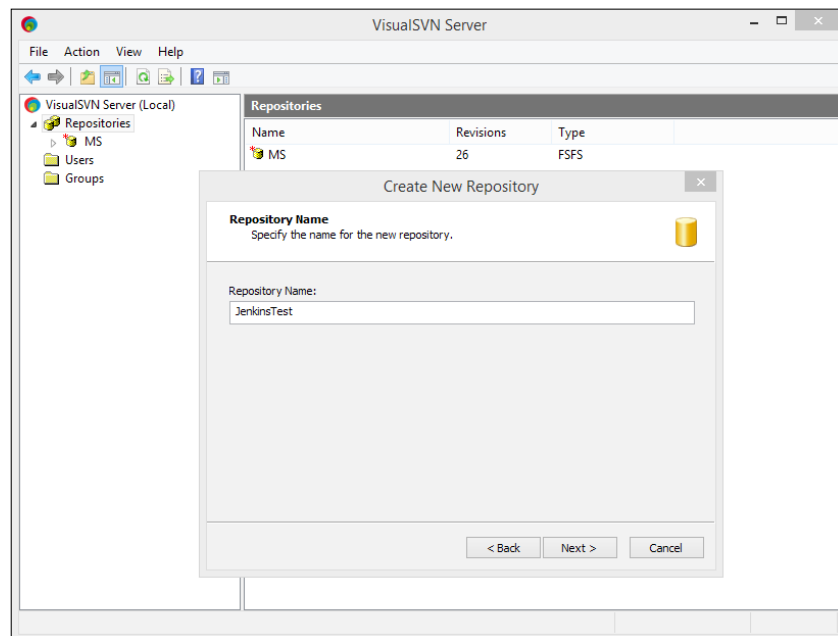
1. Download the VisualSVN server from: <https://www.visualsvn.com/server/download/>. It allows you to install and manage a fully-functional Subversion server with Windows.
2. Execute `VisualSVN-Server-x.x.x-x64.msi` and follow the wizard to install VisualSVN Server.
3. Open VisualSVN Server Manager.
4. Create a new repository, `JenkinsTest`.



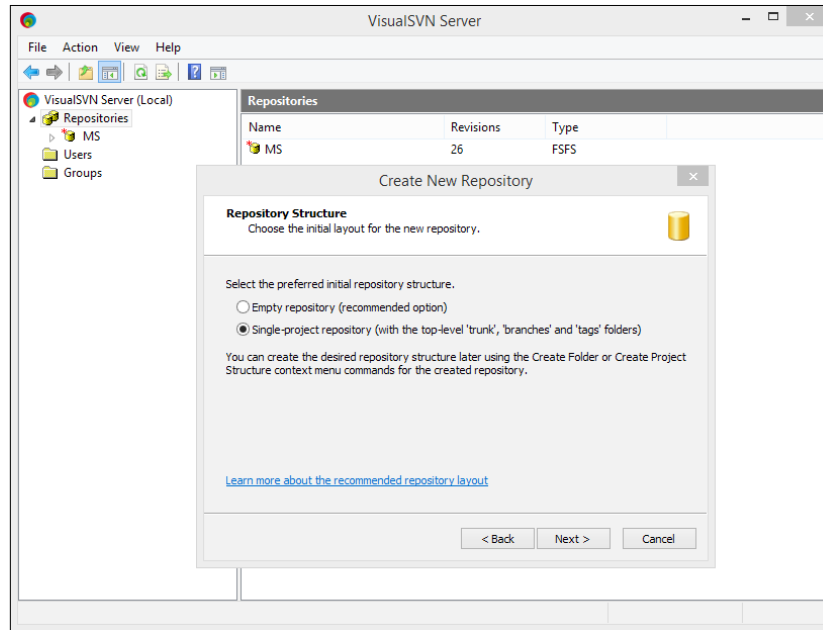
5. Select the regular subversion repository and click on **Next >**.



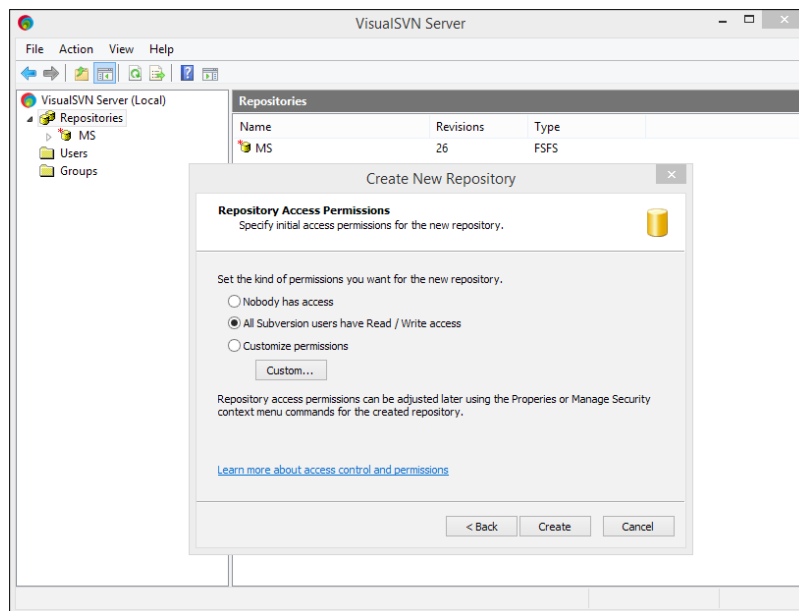
6. Provide the **Repository Name** and click on **Next >**.



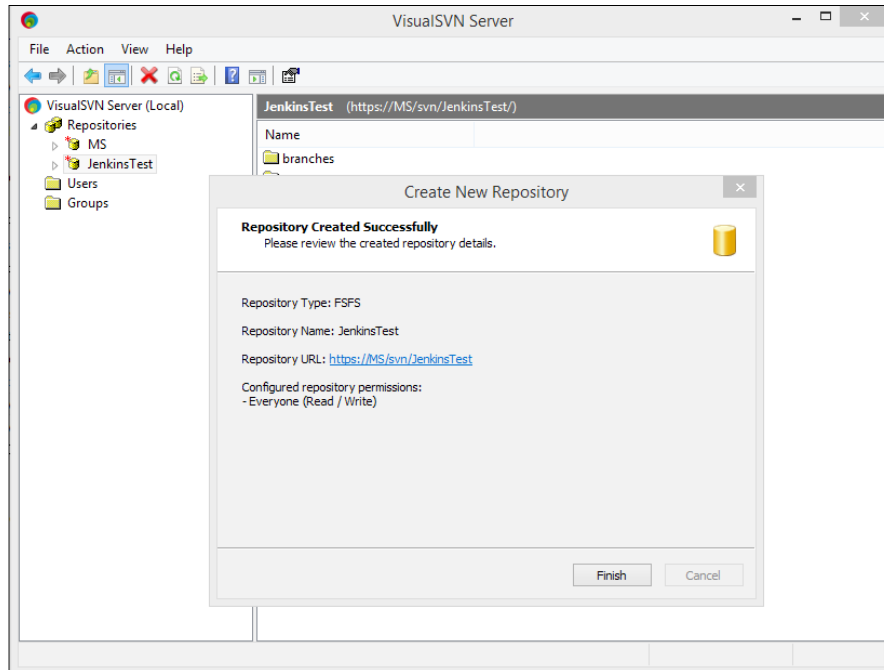
7. Select **Single-project repository** and click on **>**.



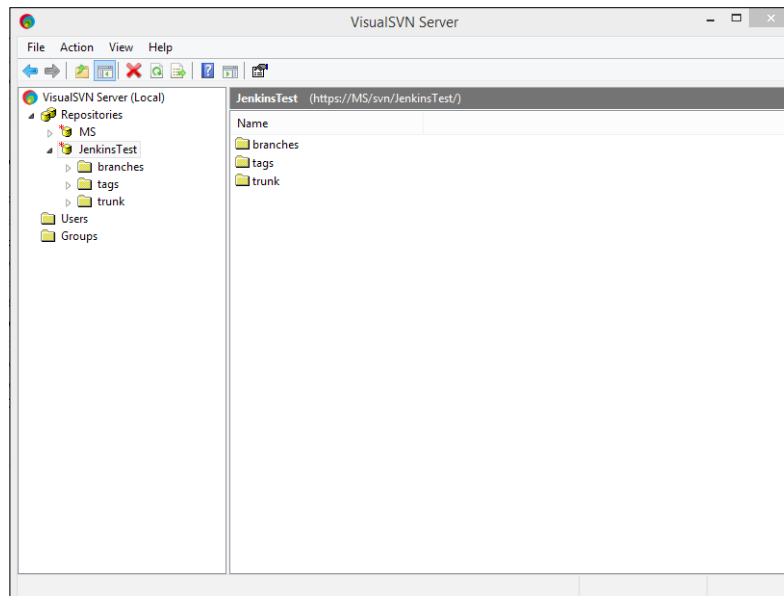
8. Select the Repository Access Permissions based on your requirements and click on **Create**.



9. Review the created repository details and click on **Finish**.



10. Verify the newly created repository in VisualSVN Server Manager.



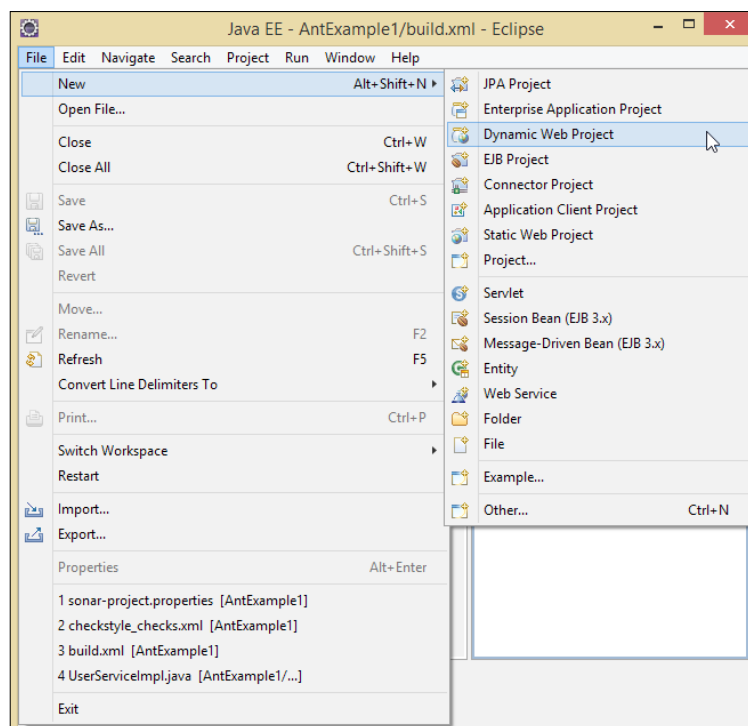
11. Verify the repository location in the browser, as shown in the following screenshot:



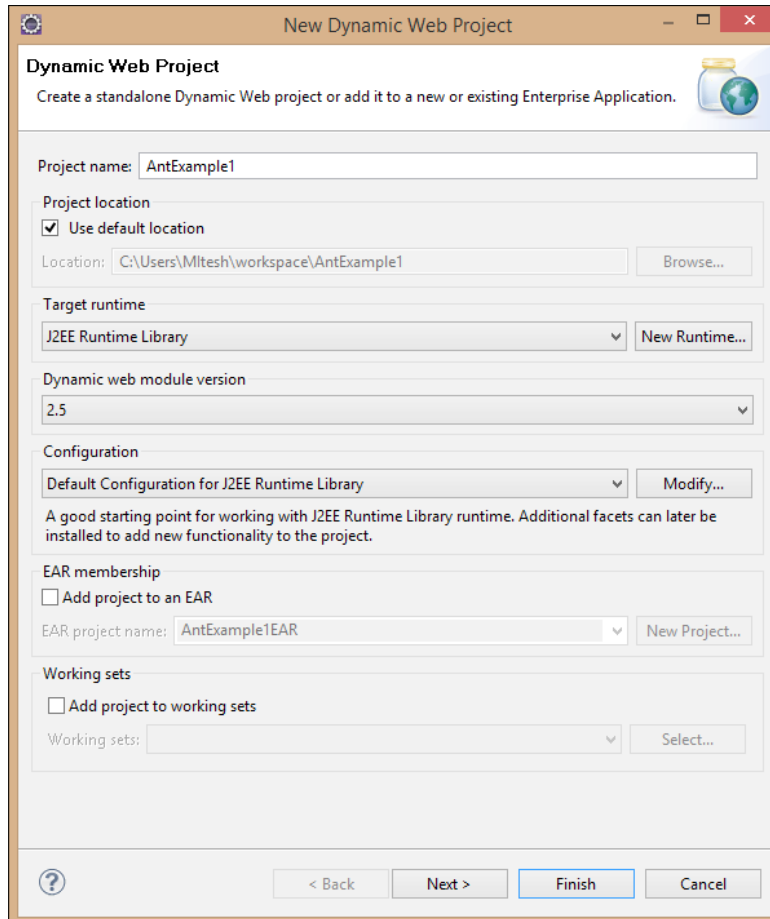
12. Now install SVN client from: <http://sourceforge.net/projects/tortoisesvn/>, to perform SVN operations.

Let's create a sample JEE project in Eclipse to illustrate SVN and Eclipse integration.

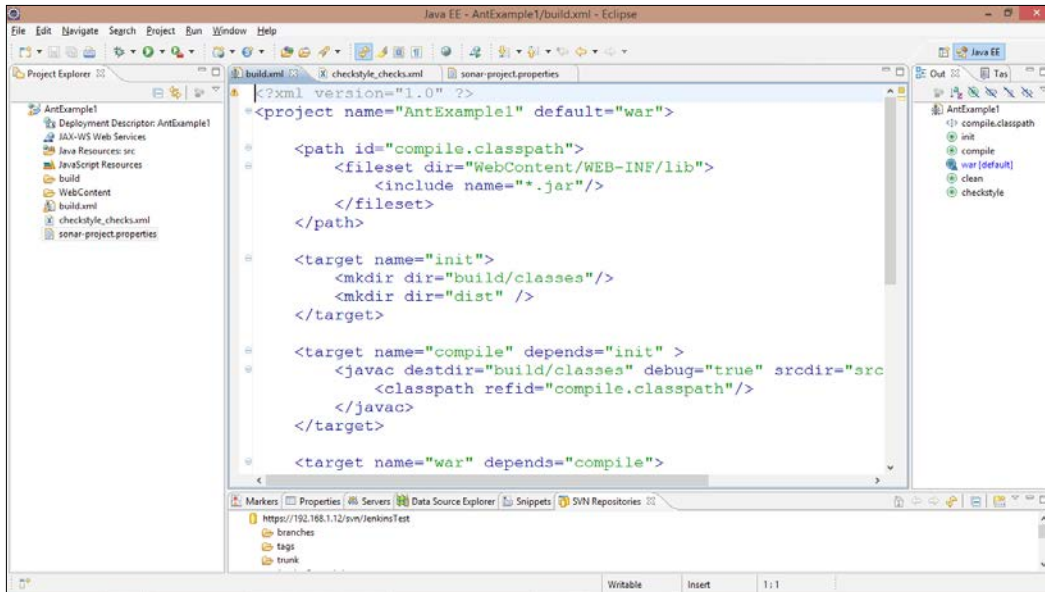
1. Open Eclipse, go to the **File** menu and click on **Dynamic Web Project**.



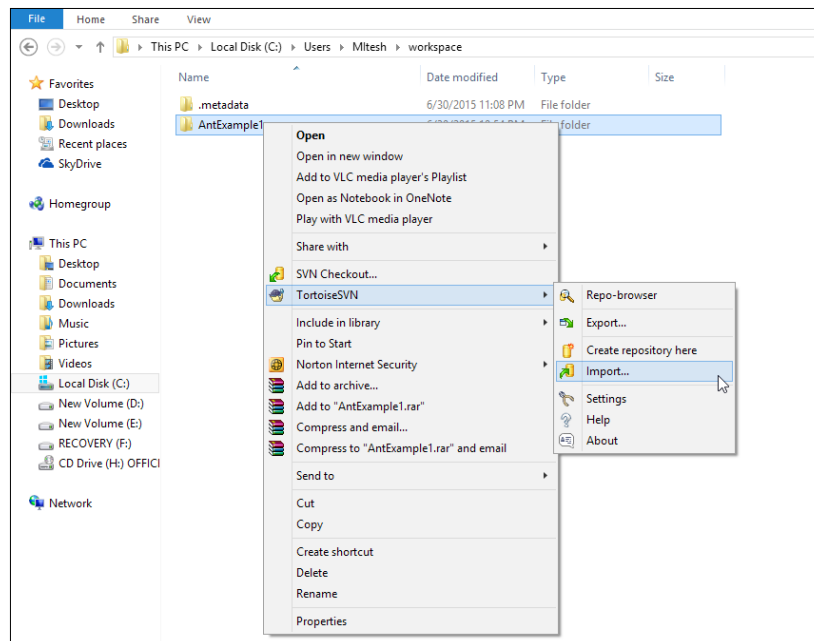
2. It will open a dialog box to create a **New Dynamic Web Project**.



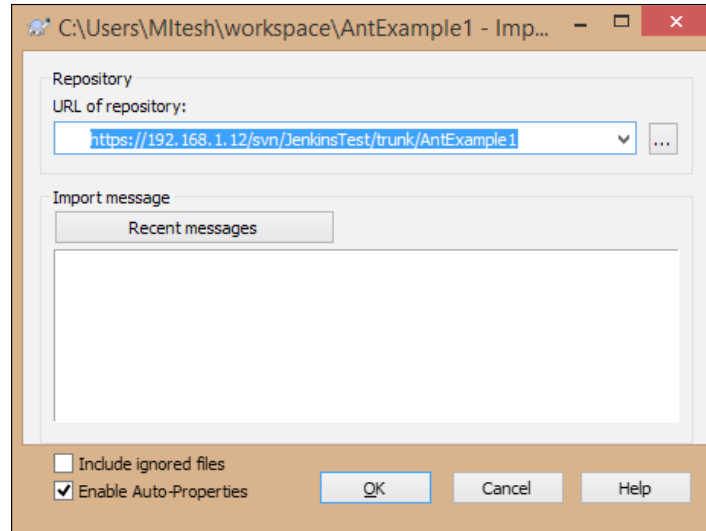
3. Create the source files and a build file for a simple project.



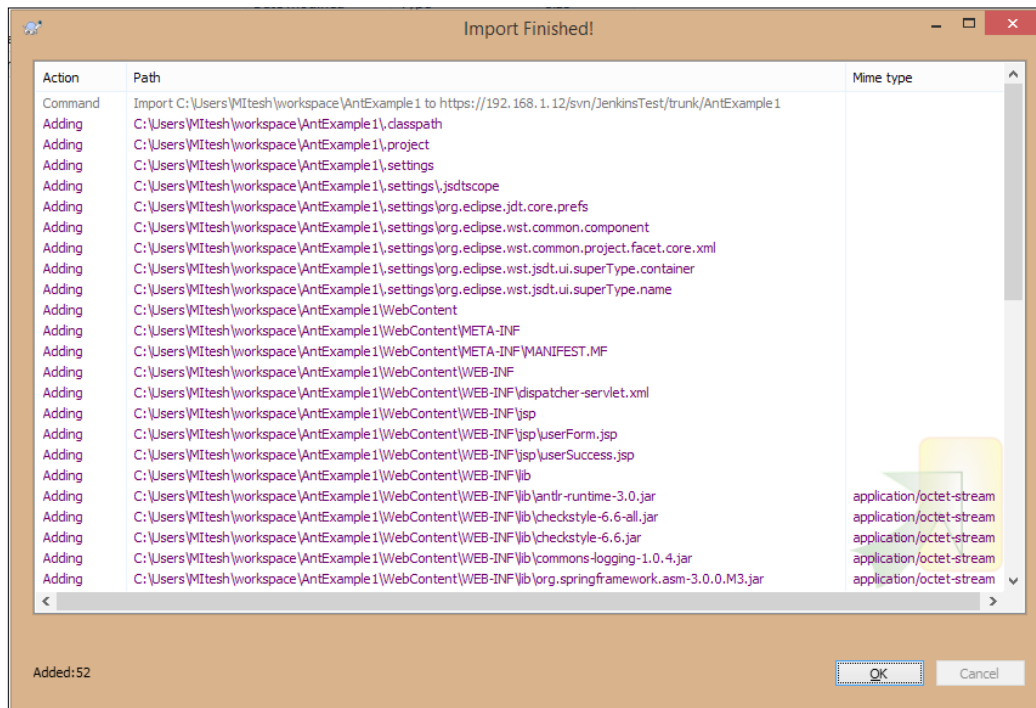
4. Go to **Application Directory**, right-click on it, select **TortoiseSVN**, and select **Import** from the sub-menu.



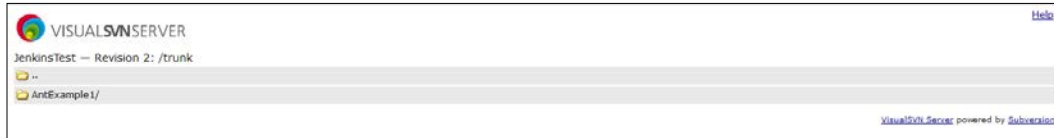
5. Enter the repository URL and click on **OK**.



6. It will add all files from the application to SVN, as shown in the following screenshot.

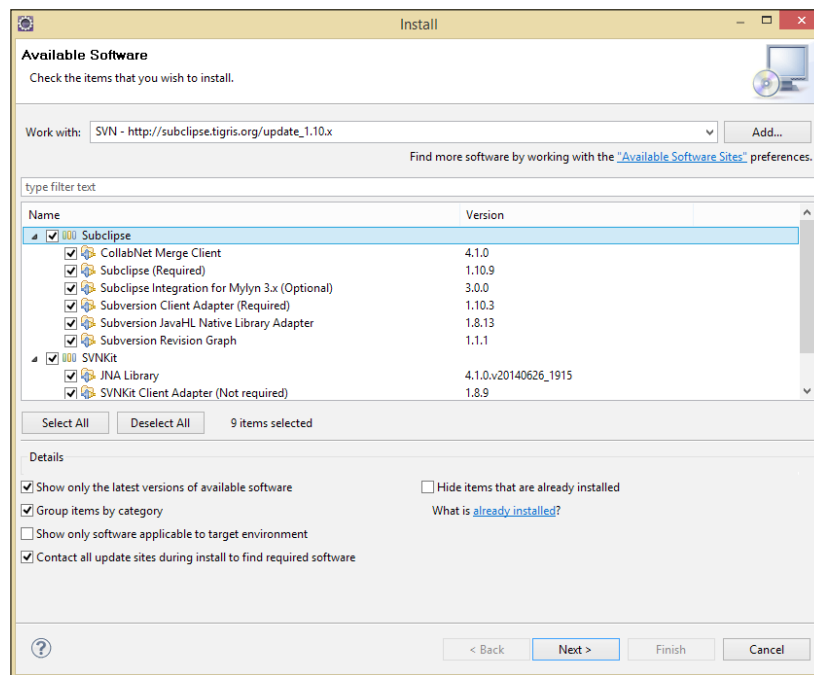


7. Verify the import by visiting the SVN repository in a browser as shown:

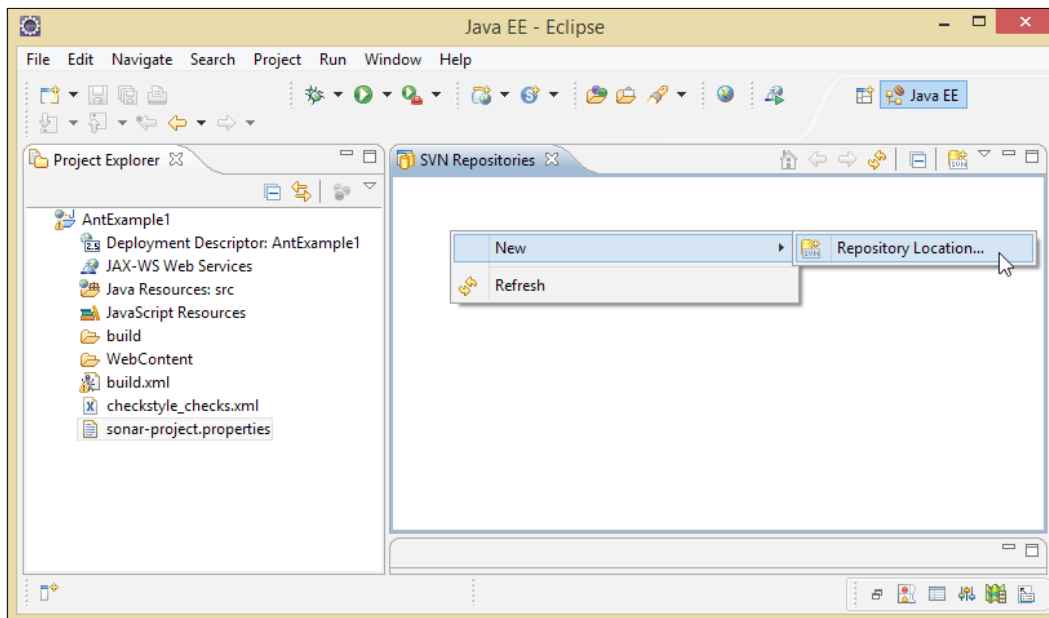


Integrating Eclipse with code repositories

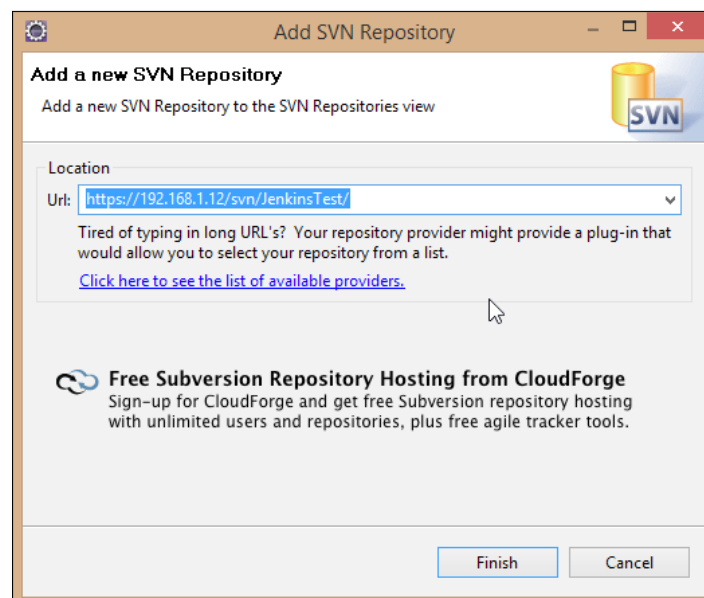
1. Open Eclipse IDE, go to the **Help** menu and click on **Install New Software**.
2. Add the repository by adding this URL: `http://subclipse.tigris.org/update_1.10.x`, then select all packages and click on **Next >**.



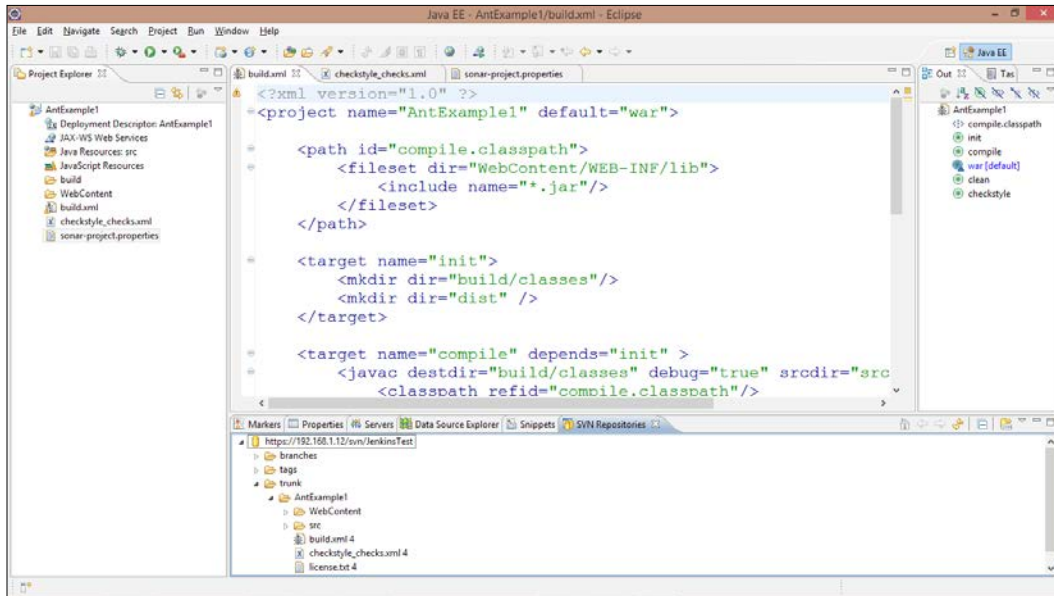
3. Review the items to be installed and the Review Licenses in the wizard. Accept the terms of agreement and click on **Finish**.
4. Restart Eclipse. Go to the **Window** menu, select **Show View**, click on **Other**, and find the SVN and SVN repositories.
5. In the SVN repositories area, right-click and select **New**; select **Repository Location...** from the sub-menu.



6. Add a new SVN Repository in Eclipse with this URL:
`https://<Ip address/ localhost / hostname>/svn/JenkinsTest/`.
7. Click on **Finish**.



8. Verify the SVN repository.



Try to integrate SVN, installed on CentOS, with Eclipse IDE, as practice.

Installing and configuring Ant

1. Download the Ant distribution from: <https://ant.apache.org/bindownload.cgi> and unzip it.
2. Set the ANT_HOME and JAVA_HOME environment variables.

```

root@localhost:~/git/development/AntExample1
File Edit View Search Terminal Tabs Help

root@localhost:~/git/development/AntExa... X root@localhost:usr X

[root@localhost Desktop]# git --version
git version 1.7.1
[root@localhost Desktop]# git config --global user.name "Mitesh"
[root@localhost Desktop]# git config --global user.email "[redacted]@gmail.com"
[root@localhost Desktop]# git config --list
user.name=Mitesh
user.email=[redacted]@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
[root@localhost Desktop]# mkdir -p ~/git/development ; cd ~/git/development
[root@localhost development]# cd AntExample1/
[root@localhost AntExample1]# git init
Initialized empty Git repository in /root/git/development/AntExample1/.git/
[root@localhost AntExample1]# git add .
[root@localhost AntExample1]#

```

There is an option available in Jenkins to install Ant or Maven automatically. We will study this in the *Configuring Ant, Maven, and JDK in Jenkins* section.

Installing Maven

Download the Maven binary ZIP file from <https://maven.apache.org/download.cgi> and extract it to the local system where Jenkins is installed.

The screenshot shows the Apache Maven Project website for version 3.3.3. The page includes a sidebar with navigation links (Main, Welcome, License, Download, Install, Configure, Run, IDE Integration, About Maven, What is Maven?, Features, FAQ, Support and Training, Documentation, Maven Plugins). The main content area is titled "Downloading Apache Maven 3.3.3" and states that this is the latest release. It provides a list of download mirrors, with the currently selected one being <http://mirror.reverse.net/pub/apache/>. Below this, there is a section for "System Requirements" which lists the following details:

Java Development Kit (JDK)	Maven 3.3 requires JDK 1.7 or above to execute - it still allows you to build against 1.3 and other JDK versions by Using Toolchains
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, additional disk space will be used for your local Maven repository. The size of your local repository will vary depending on usage but expect at least 500MB.
Operating System	No minimum requirement. Start up scripts are included as shell scripts and Windows batch files.

Configuring Ant, Maven, and JDK in Jenkins

1. Open the Jenkins dashboard in your browser with this URL: `http://<ip_address>:8080/configure`. Go to the **Manage Jenkins** section and click on **Configure System**.
2. Configure Java, based on the installation shown in the following screenshot:

The screenshot displays the 'Configure System' page in Jenkins, specifically the 'JDK installations' and 'Git' sections.

JDK installations:

- JDK 1.7:** Name is 'java1.7', JAVA_HOME is '/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.71.x86_64'. There is an unchecked checkbox for 'Install automatically' and a red 'Delete JDK' button.
- JDK 8:** Name is 'java 8', JAVA_HOME is '/opt/jdk1.8.0_45'. There is an unchecked checkbox for 'Install automatically' and a red 'Delete JDK' button.
- An 'Add JDK' button is located below the list.
- A note states: 'List of JDK installations on this system'.

Git:

- Git installations:** Name is 'Default', Path to Git executable is 'git'. There is an unchecked checkbox for 'Install automatically' and a red 'Delete Git' button.

At the bottom of the page, there are 'Save' and 'Apply' buttons.

3. Configure or install Ant automatically on the same page. Configure Maven as well.

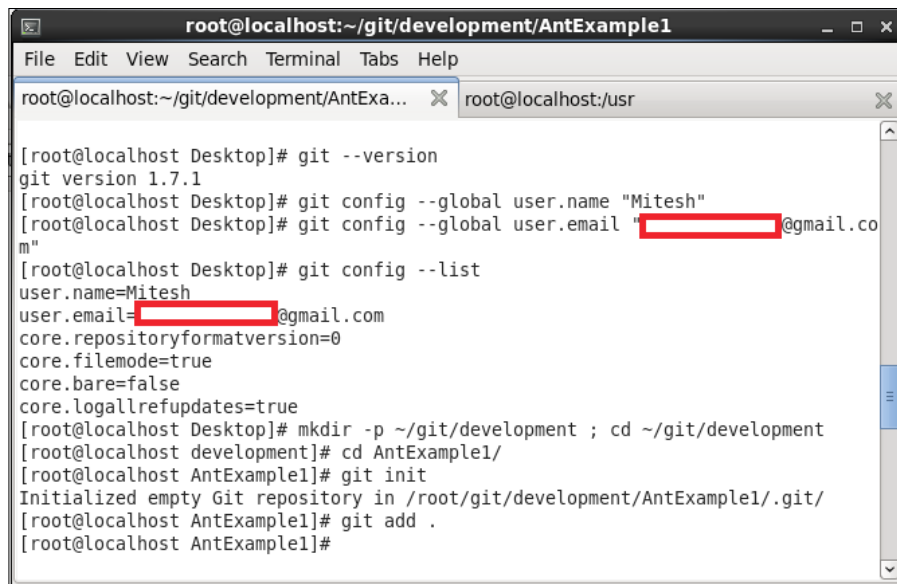
The screenshot shows a web-based configuration interface for Ant and Maven. The top section is titled 'Ant' and contains a list of 'Ant installations'. There is a single entry with 'Name' 'Ant1.9.4', 'Install automatically' checked, and 'Install from Apache' with 'Version' '1.9.4'. To the right of this entry are two red buttons: 'Delete Installer' and 'Delete Ant'. Below the list are three buttons: 'Add Installer' (with a dropdown arrow), 'Add Ant', and a small text link 'List of Ant installations on this system'. The bottom section is titled 'Maven' and contains a list of 'Maven installations'. There is a single entry with 'Name' 'Maven1.3', 'MAVEN_HOME' '/usr/lib/apache-maven-3.2.1', and 'Install automatically' unchecked. To the right of this entry is a red button 'Delete Maven'. At the bottom left of the interface are two buttons: 'Save' and 'Apply'.

Installing and configuring Git

Git is a free and open source distributed version control system. In this section, we will try to install and configure Git.

1. Open the terminal in the CentOS-based system and execute the command `yum install git` in the terminal.
2. Once it is successfully installed, verify the version with the command `git --version`.
3. Provide information about the user with the `git config` command so that commit messages will be generated with the correct information attached.

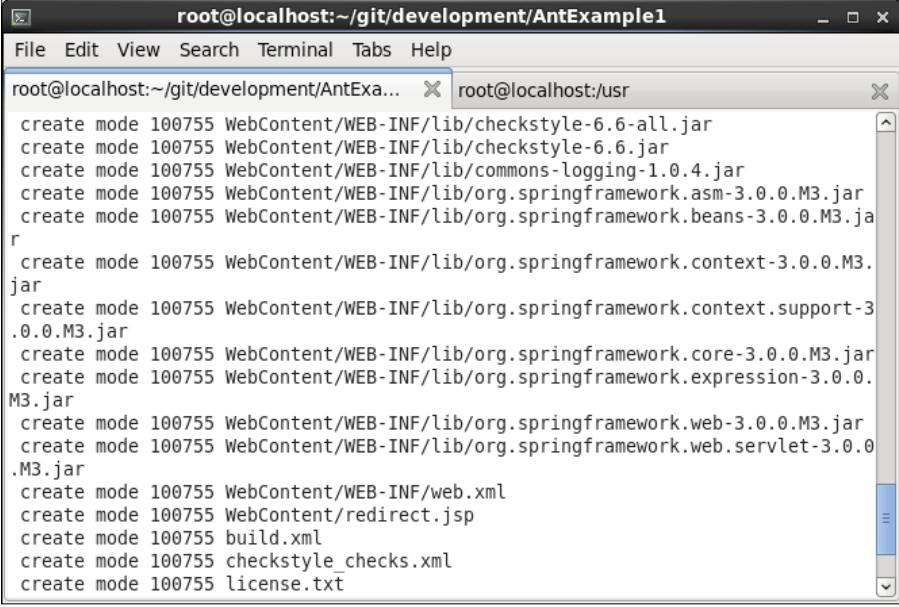
4. Provide the name and e-mail address to embed into commits.
5. To create a workspace environment, create a directory called `git` in the home directory and then create a subdirectory inside of that called `development`.
Use `mkdir -p ~/git/development ; cd ~/git/development` in the terminal.
6. Copy the `AntExample1` directory into the `development` folder.
7. Convert an existing project into a workspace environment by using the `git init` command.
8. Once the repository is initialized, add files and folders.



```
root@localhost:~/git/development/AntExample1
File Edit View Search Terminal Tabs Help
root@localhost:~/git/development/AntExa... X root@localhost:usr X

[root@localhost Desktop]# git --version
git version 1.7.1
[root@localhost Desktop]# git config --global user.name "Mitesh"
[root@localhost Desktop]# git config --global user.email "[REDACTED]@gmail.com"
[root@localhost Desktop]# git config --list
user.name=Mitesh
user.email=[REDACTED]@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
[root@localhost Desktop]# mkdir -p ~/git/development ; cd ~/git/development
[root@localhost development]# cd AntExample1/
[root@localhost AntExample1]# git init
Initialized empty Git repository in /root/git/development/AntExample1/.git/
[root@localhost AntExample1]# git add .
[root@localhost AntExample1]#
```


9. Commit by executing `git commit -m "Initial Commit" -a`.

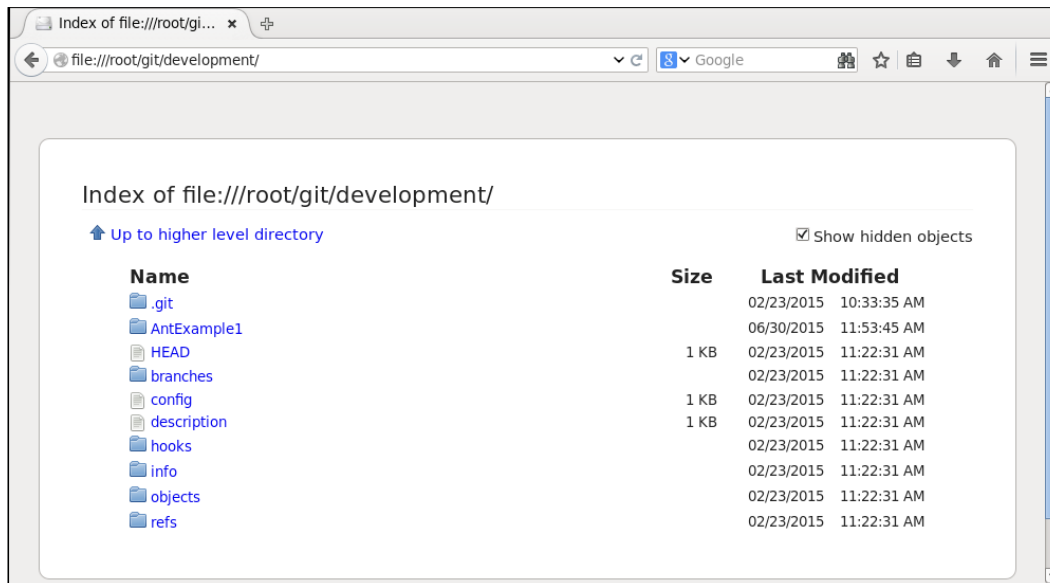


```

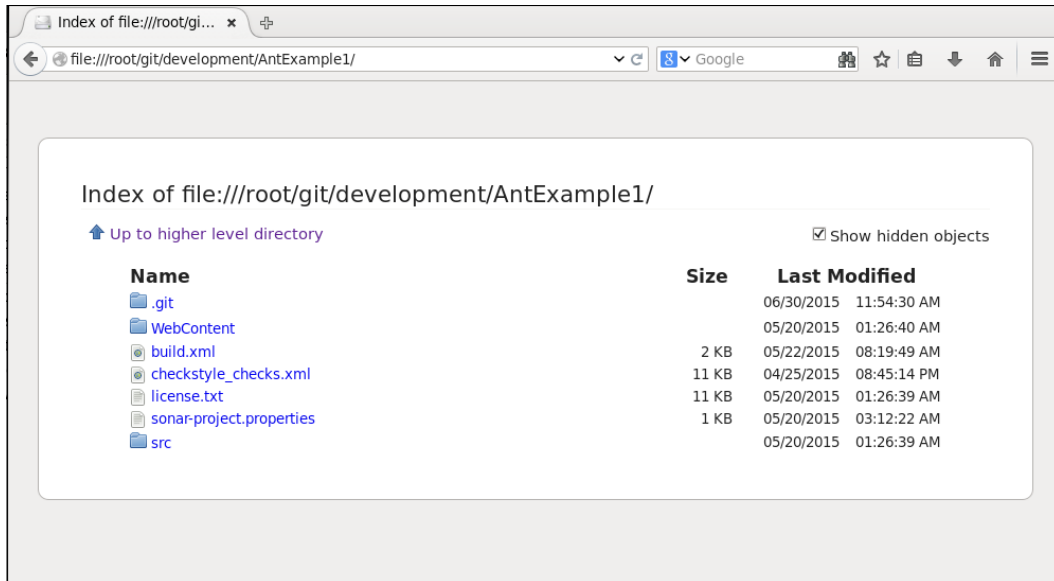
root@localhost:~/git/development/AntExample1
File Edit View Search Terminal Tabs Help
root@localhost:~/git/development/AntExa... X root@localhost:usr X
create mode 100755 WebContent/WEB-INF/lib/checkstyle-6.6-all.jar
create mode 100755 WebContent/WEB-INF/lib/checkstyle-6.6.jar
create mode 100755 WebContent/WEB-INF/lib/commons-logging-1.0.4.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.asm-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.beans-3.0.0.M3.jar
r
create mode 100755 WebContent/WEB-INF/lib/org.springframework.context-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.context.support-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.core-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.expression-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.web-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/lib/org.springframework.web.servlet-3.0.0.M3.jar
create mode 100755 WebContent/WEB-INF/web.xml
create mode 100755 WebContent/redirect.jsp
create mode 100755 build.xml
create mode 100755 checkstyle checks.xml
create mode 100755 license.txt

```

10. Verify the Git repository



11. Verify the project in the Git repository.

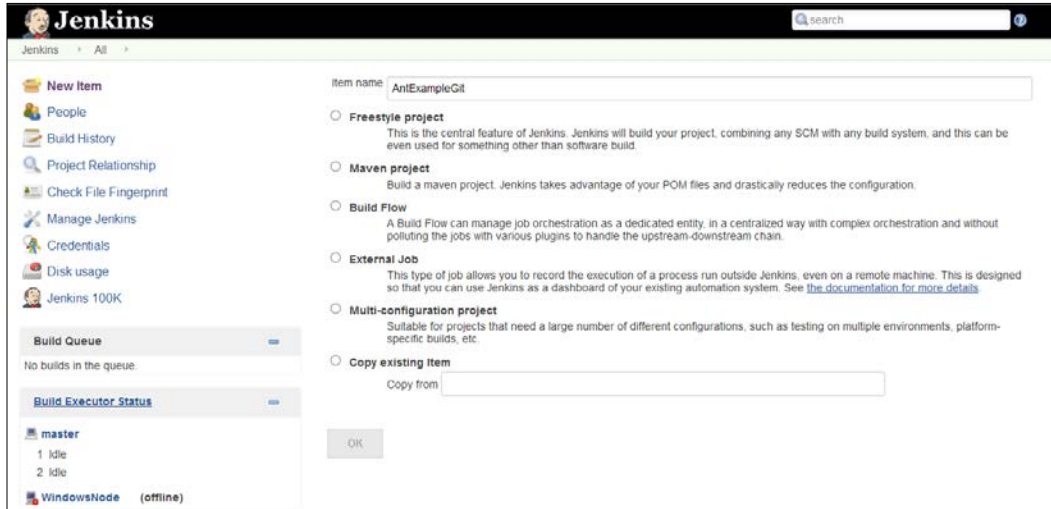


Creating a new build job in Jenkins with Git

1. On the Jenkins dashboard, click on **Manage Jenkins** and select **Manage Plugins**. Click on the **Available** tab and write `github` plugin in the search box.
2. Click the checkbox and click on the button, **Download now and install after restart**.
3. Restart Jenkins.

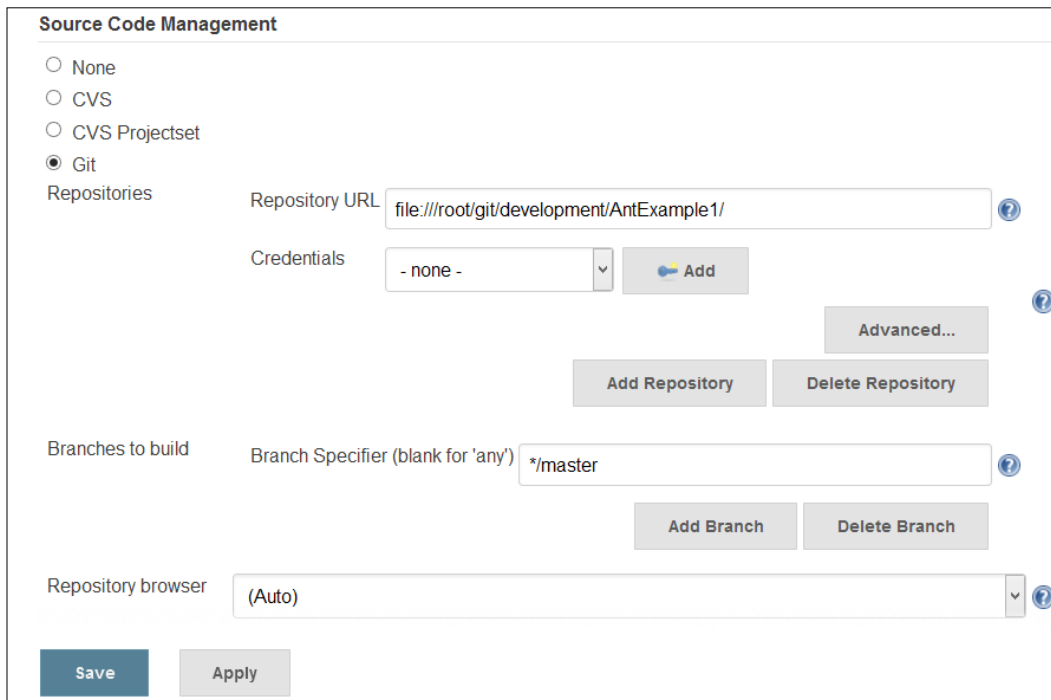


4. Create a new **Freestyle project**. Provide **Item name** and click on **OK**.



The screenshot shows the Jenkins 'New Item' configuration page. The 'Item name' field is filled with 'AntExampleGit'. Under the 'Select which SCM tool to use' section, the 'Freestyle project' radio button is selected. The left sidebar shows the 'Build Queue' and 'Build Executor Status' sections. The 'Build Queue' shows 'No builds in the queue'. The 'Build Executor Status' shows two idle executors on the 'master' branch and one offline executor on the 'WindowsNode'.

5. Configure **Git** in the **Source Code Management** section.



The screenshot shows the 'Source Code Management' configuration section in Jenkins. The 'Git' radio button is selected under 'None', 'CVS', 'CVS Projectset', and 'Git'. The 'Repository URL' field is filled with 'file:///root/git/development/AntExample1/'. The 'Credentials' dropdown is set to '- none -'. The 'Branches to build' section has the 'Branch Specifier (blank for 'any')' field filled with '*/master'. The 'Repository browser' dropdown is set to '(Auto)'. The 'Save' and 'Apply' buttons are at the bottom.

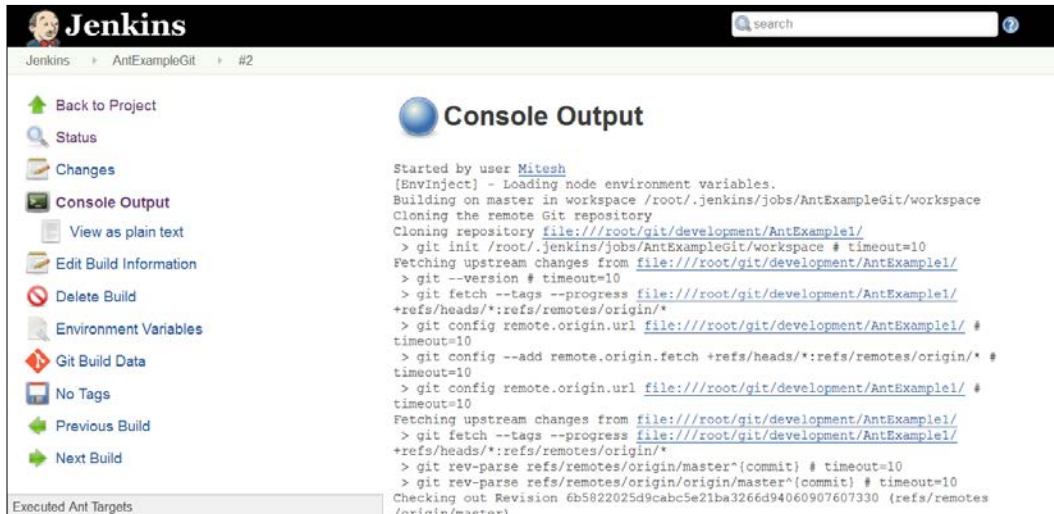
6. Add the **Invoke Ant** build step by clicking on **Add build step**.

The screenshot shows the 'Build' configuration page in Jenkins. Under the 'Build' section, the 'Invoke Ant' step is selected. The 'Ant Version' is set to 'Default' and the 'Targets' are set to 'war'. There are 'Advanced...' and 'Delete' buttons. Below this, there is an 'Add build step' button. Under the 'Post-build Actions' section, 'Publish combined analysis results' is selected, with checkboxes for 'Checkstyle warnings', 'FindBugs warnings', and 'Compiler warnings', all of which are checked. At the bottom, there are 'Save' and 'Apply' buttons.

7. Execute the build.

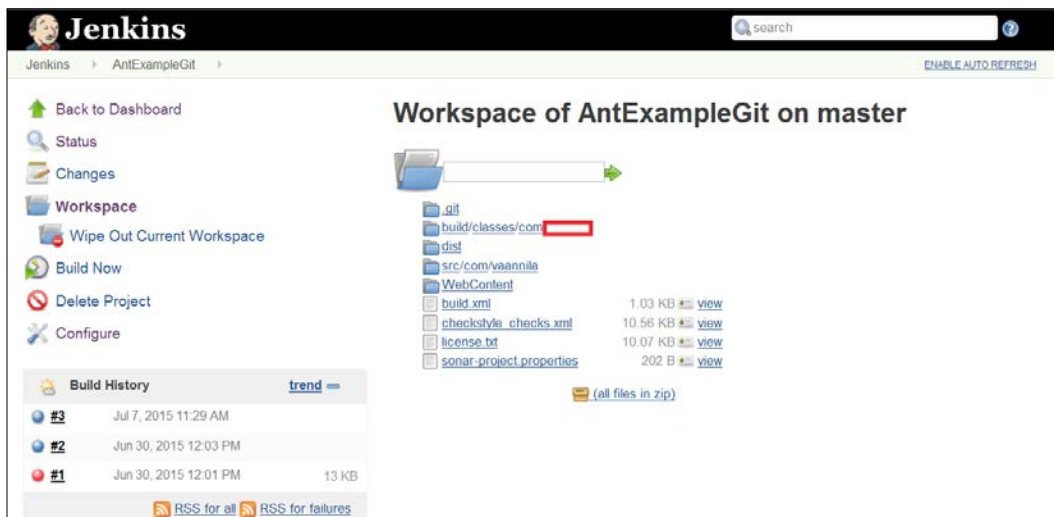
The screenshot shows the Jenkins 'Build #2' page. The build status is 'PM)'. The build was started 10 days ago and took 6.8 sec on master. The build was started by user 'Mitesh'. The revision is 6b5822025d9cabc5e21ba3266d94060907607330, which is refs/remotes/origin/master. The page includes a sidebar with links to 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete Build', 'Environment Variables', 'Git Build Data', 'No Tags', 'Previous Build', and 'Next Build'. The footer shows 'Page generated: Jul 11, 2015 7:23:27 AM' and 'Jenkins ver. 1.606'.

8. Click on **Console Output** to see the progress of the build.



The screenshot shows the Jenkins web interface for a job named 'AntExampleGit'. The 'Console Output' tab is selected, displaying a log of the build process. The log starts with 'Started by user Mitesh' and shows the execution of 'mvn clean install' commands. The build is successful, with the final output being 'Checking out Revision 6b5822025d9cabc5e21ba3266d94060907607330 (refs/remotes/origin/master)'.

9. Once the build has succeeded, verify **Workspace** in the build job.



The screenshot shows the Jenkins web interface for the same job 'AntExampleGit'. The 'Workspace' tab is selected, displaying the workspace of 'AntExampleGit on master'. The workspace contains several files and directories, including '.git', 'build/classes/com', 'dist', 'src/com/vaannila', 'WebContent', 'build.xml', 'checkstyle_checks.xml', 'license.txt', and 'sonar-project.properties'. The 'Build History' section shows three builds: #3 (Jul 7, 2015 11:29 AM), #2 (Jun 30, 2015 12:03 PM), and #1 (Jun 30, 2015 12:01 PM, 13 KB).

10. Done!

Self-test questions

Q1. Where to set the JAVA_HOME and JRE_HOME environment variables?

1. /root/ .bash_profile
2. /root/ .env_profile
3. /root/ .bash_variables
4. /root/ .env_variables

Q2. Which are valid SVN operations?

1. `svn import /tmp/mytestproj/`
2. `svn co http://localhost/repos/mytestproj`
3. Both the above

Q3. Where do you configure Java and Ant in Jenkins?

1. Go to the **Manage Jenkins** section and click on **Configure System**
2. Go to the **Manage Jenkins** section and click on **Global Configuration**

Summary

Hooray! We have reached the end of this chapter. We have covered how to prepare an environment for continuous integration by setting up a local CentOS repository, installing code repositories such as SVN on CentOS and Windows, and build tool Ant. We have also seen detailed instructions on how to configure repositories and build tools in Jenkins. Finally, we have covered how to integrate the Integrated Development Environment with code repositories so that efficient development and ease of `commit` operations can take place to facilitate the deployment pipeline process.

3

Integration of Jenkins, SVN, and Build Tools

"The barrier to change is not too little caring; it is too much complexity"

– Bill Gates

We have seen how to set up an environment to use Jenkins for continuous integration, and we have also configured build tools in Jenkins. The integration of Eclipse with SVN will help developers to easily perform operations on repositories.

Now we are ready to create our first build job for continuous integration.

This chapter describes in detail how to create and configure build jobs for Java applications using build tools such as Ant and Maven; how to run build jobs, unit test cases. It covers all aspects of running a build to create a distribution file or war file for deployment, as well as a Dashboard View plugin to provide a customized display of build jobs and test results based on preferences. The following are the main points which are covered in this chapter:

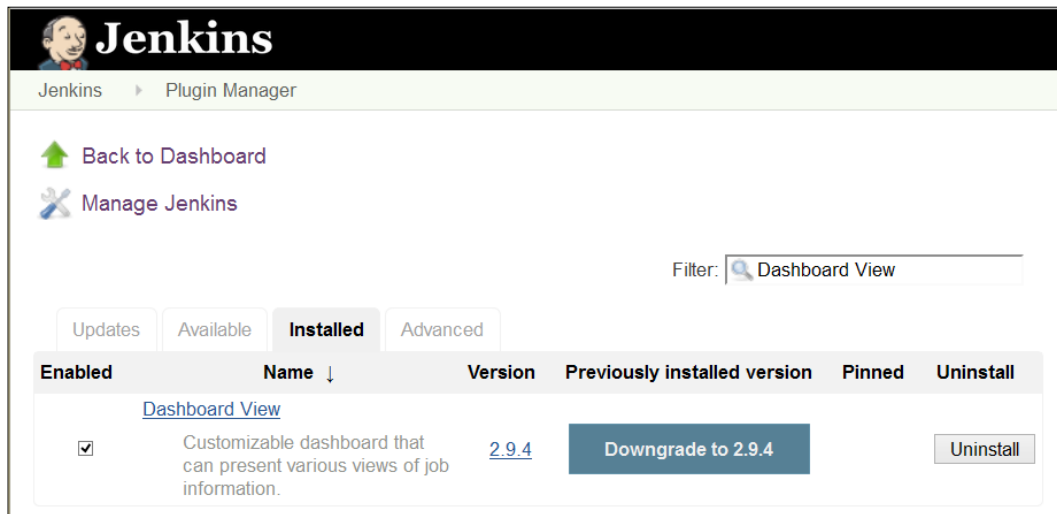
- Creating and configuring a build job for a Java application with Ant
- Creating and configuring a build job for a Java application with Maven
- Build execution with test cases

Creating and configuring a build job for a Java application with Ant

Before creating and configuring a build job for a Java application, we will install a Dashboard View plugin to better manage builds, and display the results of builds and tests. We have already seen how to create a basic job in *Chapter 2, Installation and Configuration of Code Repository and Build Tools*.

Dashboard View Plugin

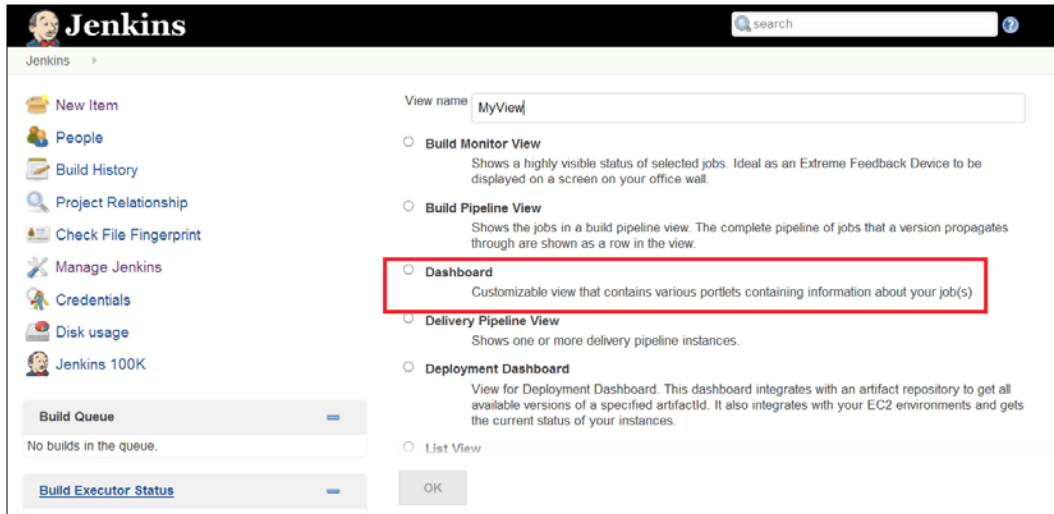
This plugin presents a new view that provides a portal-like view for Jenkins build jobs. Download it from <https://wiki.jenkins-ci.org/display/JENKINS/Dashboard+View>. It is good for showing results and trends. In addition, it also allows the user to arrange display items in an effective manner. On the Jenkins dashboard, go to the **Manage Jenkins** link and click on **Manage Plugins** and install the Dashboard View plugin. Verify the installation by clicking on the **Installed** tab.



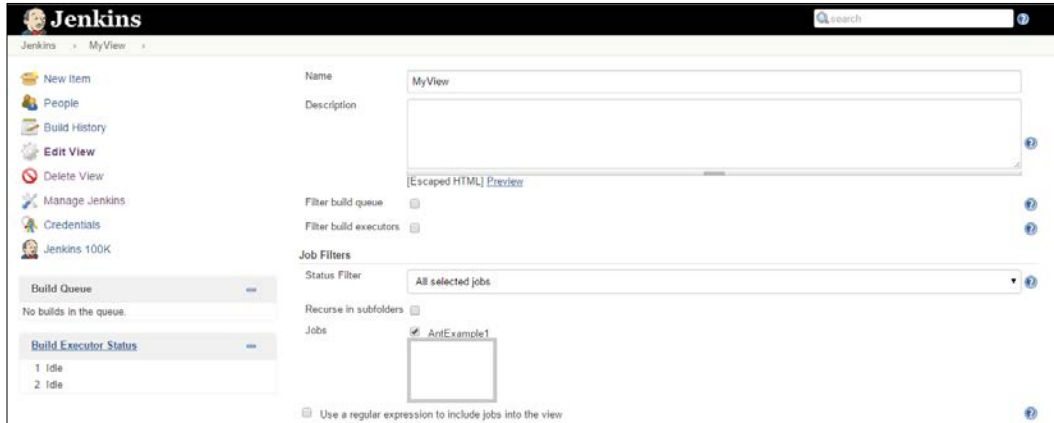
The screenshot shows the Jenkins Plugin Manager interface. At the top, there's a header with the Jenkins logo and the text "Jenkins". Below it, a breadcrumb trail shows "Jenkins" and "Plugin Manager". On the left, there are two links: "Back to Dashboard" with a green arrow icon and "Manage Jenkins" with a wrench icon. On the right, there's a search filter labeled "Filter:" with a magnifying glass icon and the text "Dashboard View". Below the search bar, there are four tabs: "Updates", "Available", "Installed" (which is selected), and "Advanced". Under the "Installed" tab, there's a table with the following columns: "Enabled", "Name", "Version", "Previously installed version", "Pinned", and "Uninstall". The table contains one entry for the "Dashboard View" plugin. The "Enabled" column has a checked checkbox. The "Name" column shows the link "Dashboard View". The "Version" column shows "2.9.4". The "Previously installed version" column is empty. The "Pinned" column is empty. The "Uninstall" column has a button labeled "Uninstall".

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
☒	Dashboard View Customizable dashboard that can present various views of job information.	2.9.4			Downgrade to 2.9.4 Uninstall

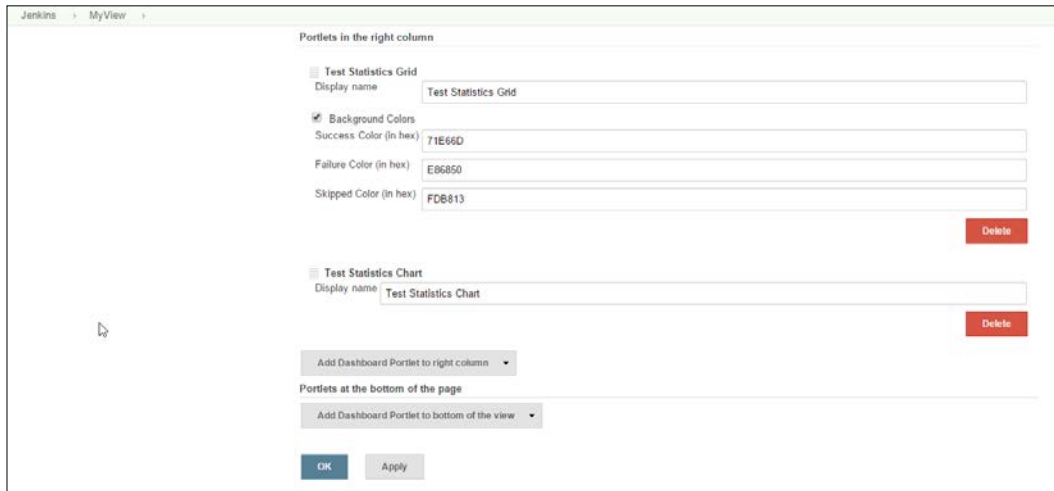
On the Jenkins dashboard, click on the plus button to create a new view. Provide a **View name** and select the type of view; in our case **Dashboard**, then click on **OK**.



Provide a **Name** and select **Jobs** that need to be included in the view, as shown in the following screenshot:



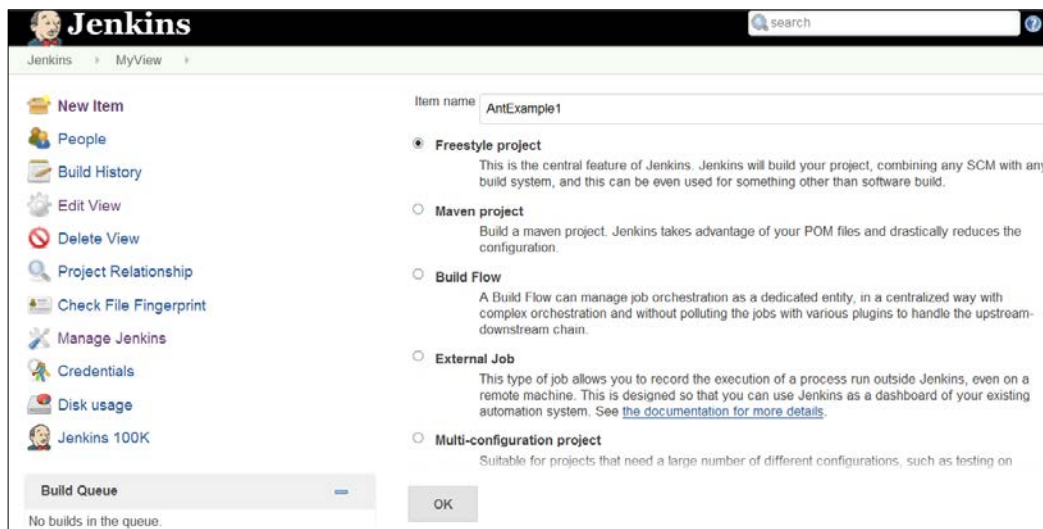
In the View configuration, click on **Add Dashboard Portlet to right column**, and select **Test Statistics Grid**. Add **Test Statistics Chart**. This will display test results in the form of statistics and chart representations of test results.



The screenshot shows the Jenkins 'MyView' configuration page. Under 'Portlets in the right column', there are two sections: 'Test Statistics Grid' and 'Test Statistics Chart'. The 'Test Statistics Grid' section has a 'Display name' field set to 'Test Statistics Grid' and a checked 'Background Colors' checkbox. Below this are three color selection fields: 'Success Color (in hex)' set to '71E66D', 'Failure Color (in hex)' set to 'E8685D', and 'Skipped Color (in hex)' set to 'FDB813'. Each section has a 'Delete' button. Below these sections is a dropdown menu 'Add Dashboard Portlet to right column'. Under 'Portlets at the bottom of the page', there is a dropdown menu 'Add Dashboard Portlet to bottom of the view'. At the bottom are 'OK' and 'Apply' buttons.

Creating and configuring a build job for a Java application

Click on **New Item** on the dashboard to create a new build for a Java application which uses Ant as a build tool. Enter **Item name**, and select **Freestyle project**. Click OK.



The screenshot shows the Jenkins 'New Item' configuration page. On the left is a sidebar with navigation links: 'New Item', 'People', 'Build History', 'Edit View', 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'Credentials', 'Disk usage', and 'Jenkins 100K'. Below these is a 'Build Queue' section showing 'No builds in the queue.' The main area has an 'Item name' field set to 'AntExample1'. Below this are five radio button options: 'Freestyle project' (selected), 'Maven project', 'Build Flow', 'External Job', and 'Multi-configuration project'. Each option has a brief description. At the bottom right is an 'OK' button.

It will open the configuration for a new build job. In **Source Code Management**, select **Subversion**. Provide the **Repository URL** and **Credentials**. In *Chapter 2, Installation and Configuration of Code Repository and Build Tools*, we installed Subversion and also added the source code to SVN.

Provide the URL you use in your browser to access the source code repository.

Source Code Management

☐ None
☐ CVS
☐ CVS Projectset
☐ Git
☒ Subversion

Modules

Repository URL

https://192.168.1.12/svn/MS/AntExample1

Credentials

mitesh51/***** (SVN)

Add

Local module directory

.

Repository depth

infinity

Ignore externals

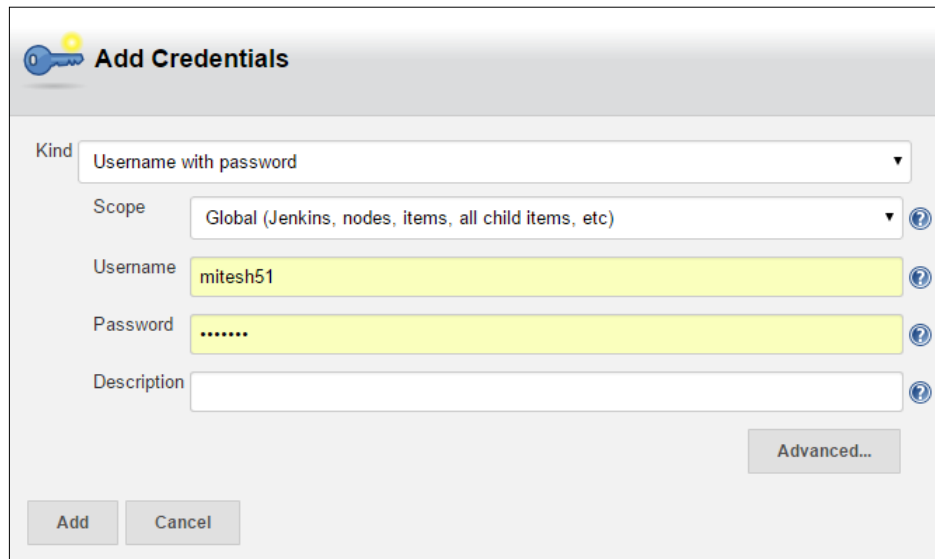
☒

Add module...

Save

Apply

If **Credentials** are not available in the box, click on the **Add** button. Provide **Scope**, **Username**, **Password**, and **Description**, and click on **Add** to make it available on the list box available in the build job configuration. **Scope** determines where credentials can be used. For example system scope restricts credential usage to the object with which the credential is associated. It provides better confidentiality than global scope. Global scope credentials are available to the object with which the credential is associated and all objects that are children of that object.



In the build job configuration, go to the **Build Triggers** section and select the **Poll SCM** radio button. Provide the schedule detail in the * * * * * form, as shown in the following figure. It will poll the repository every minute to verify changes committed into the repository by developers.

Build Triggers

☐ Build after other projects are built
 ☐ Build periodically
 ☐ Build when a change is pushed to GitHub
 ☒ Poll SCM

Schedule

Do you really mean "every minute" when you say "*****"? Perhaps you meant "H *****" to poll once per hour

☐ Ignore post-commit hooks

The **Schedule** field follows cron syntax, MINUTE HOUR Day Of the Month MONTH Day Of the Week.

For example, H ***** to poll once per hour, H/15 ***** to poll every fifteen minutes.

Once **Build Triggers** and **Source Code Management** configurations are completed, we need to provide build tool-related details, so Jenkins can use them to execute once the build is triggered. Click on the **Add build step** and select **Invoke Ant**. From the drop-down menu, select Ant, configured in *Chapter 2, Installation and Configuration of Code Repository and Build Tools* and provide **Targets** with the name you want to execute from the build.

Build

Invoke Ant

Ant Version

Ant1.9.4

Targets

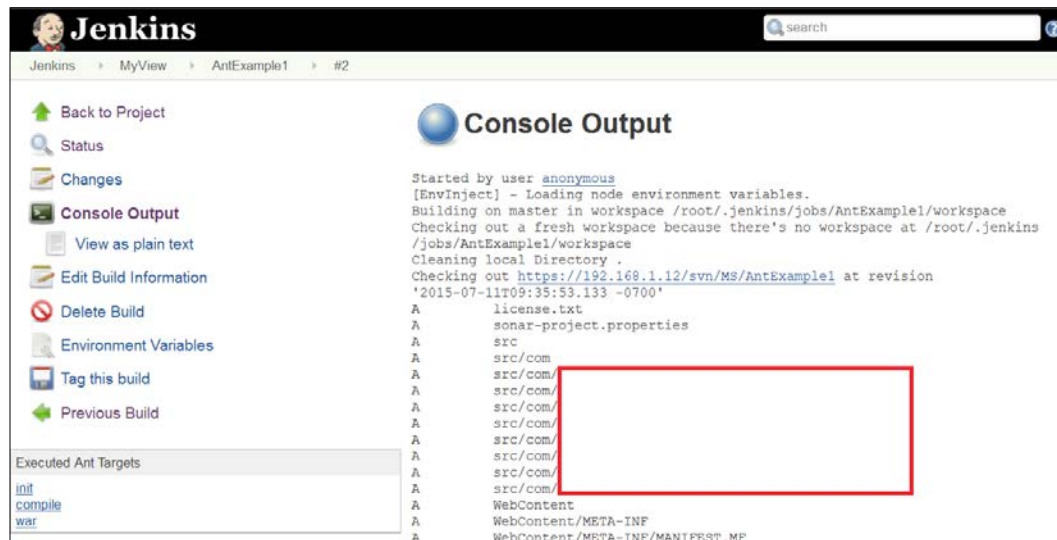
WAR

Advanced...

Delete

Add build step

Click on the **Apply** and **Save** buttons to finalize the configuration. Click on the **Build Now** button on the Jenkins dashboard. It will check out all the latest available code in the source code repository against the local workspace on the machine where Jenkins is installed, as shown in the following figure. In the **build history** section of a specific job, click on **build number**, and then click on **Console Output**.



Once the checkout process is completed, the build file execution, based on the targets, will start, and the build execution will be successful if all dependencies and files required for the build execution are available in the local workspace, as shown in the following figure:

```

Buildfile: /root/.jenkins/jobs/AntExample1/workspace/build.xml

init:
  [mkdir] Created dir: /root/.jenkins/jobs/AntExample1/workspace/build/classes
  [mkdir] Created dir: /root/.jenkins/jobs/AntExample1/workspace/dist

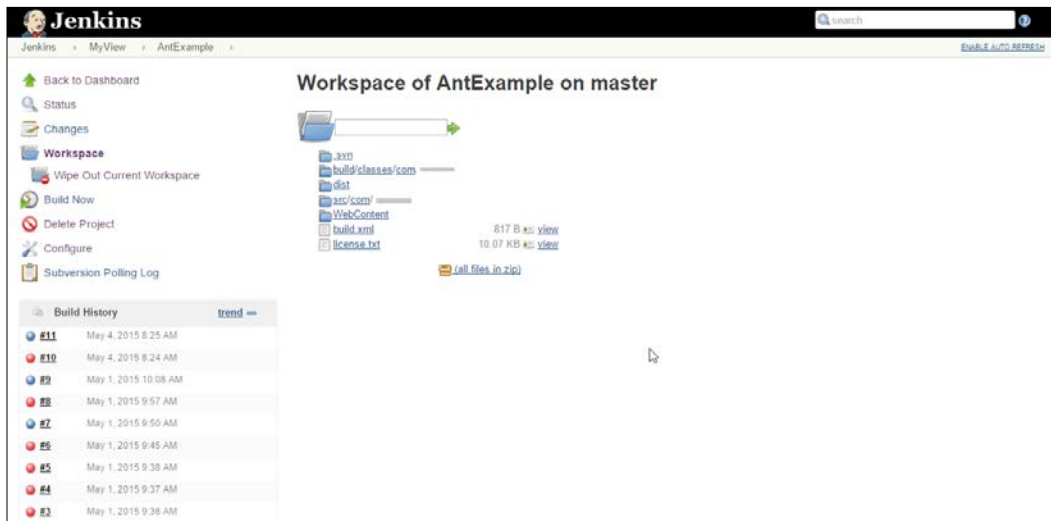
compile:
  [javac] /root/.jenkins/jobs/AntExample1/workspace/build.xml:16: warning:
  'includeantruntime' was not set, defaulting to build.sysclasspath=last; set to
  false for repeatable builds
  [javac] Compiling 4 source files to /root/.jenkins/jobs/AntExample1/workspace
  /build/classes
  [javac] Note: /root/.jenkins/jobs/AntExample1/workspace/src/com/vaannila
  /web/UserController.java uses or overrides a deprecated API.
  [javac] Note: Recompile with -Xlint:deprecation for details.

war:
  [war] Building war: /root/.jenkins/jobs/AntExample1/workspace
  /dist/AntExample.war

BUILD SUCCESSFUL
Total time: 5 seconds
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: SUCCESS

```

To verify the local workspace, go to the view you created, select **build job** and then click on **Workspace**. Verify that all files and folders are available, as provided by the source code repository.



Creating and configuring a build job for a Java application with Maven

Click on **New Item** on the dashboard to create a new build for a Java application which uses Maven as a build tool. Enter the **Item name** and select **Maven project** from the list.

The screenshot shows the Jenkins 'New Item' configuration page. The 'Item name' field is set to 'CounterApp'. The 'Maven project' option is selected under the project type list. The left sidebar shows navigation links like 'New Item', 'People', 'Build History', etc. The bottom shows 'Build Queue' and 'Build Executor Status' sections.

Item name: CounterApp

- ☐ Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- ☒ **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- ☐ Build Flow
A Build Flow can manage job orchestration as a dedicated entity, in a centralized way with complex orchestration and without polluting the jobs with various plugins to handle the upstream-downstream chain.
- ☐ External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).
- ☐ Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on

OK

It will open the configuration for the new build job. In **Source Code Management**, select **Subversion**. Provide **Repository URL** and **Credentials**. In *Chapter 2, Installation and Configuration of Code Repository and Build Tools* we installed **Subversion**, and added the source code to SVN.

Source Code Management

☐ None
☐ CVS
☐ CVS Projectset
☐ Git
☒ Subversion

Modules

Repository URL

https://192.168.1.12/svn/MS/CounterApp ?

Credentials

mitesh51/***** (SVN) Add

Local module directory

.

Repository depth

infinity

Ignore externals

☒

Add module...

Save

Apply

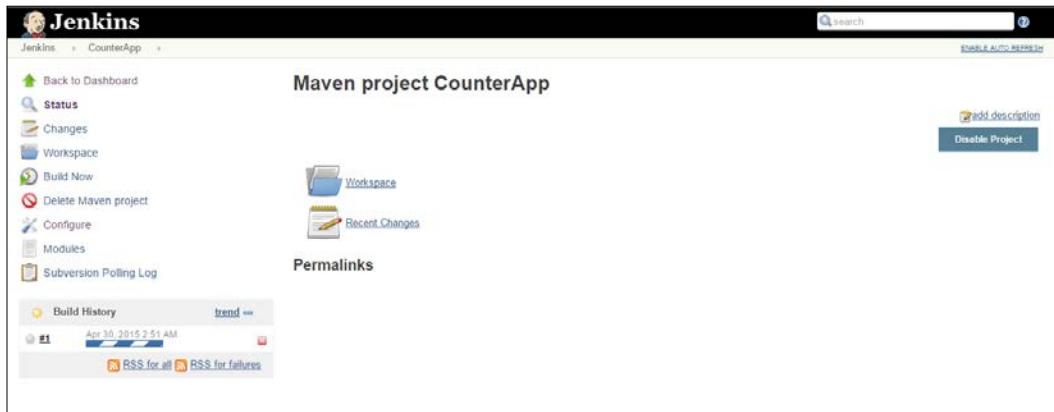
In the build job configuration, go to the **Build Triggers** section and select the **Poll SCM** radio button. Provide the schedule detail in * * * * * form, as shown in following figure. It will poll the repository every minute to verify changes committed into the repository by developers. Add the Maven build step. Provide the name of the build file; by default it is pom.xml. Provide **Goals and Options** and, if you keep it empty, then it will execute the default goal.

The screenshot displays the Jenkins build configuration interface. It is divided into three main sections: Pre Steps, Build, and Post Steps.

- Pre Steps:** Contains a step titled "Invoke top-level Maven targets". It has a "Maven Version" dropdown set to "Maven1.3" and a "Goals" text field containing "clean". There are "Advanced..." and "Delete" buttons for this step. Below this is an "Add pre-build step" button.
- Build:** Contains a "Root POM" text field with "pom.xml" and a "Goals and options" text field with "package". There is an "Advanced..." button for this section.
- Post Steps:** Contains three radio button options: "Run only if build succeeds", "Run only if build succeeds or is unstable", and "Run regardless of build result". The "Run regardless of build result" option is selected.

At the bottom, there are "Save" and "Apply" buttons.

Click on **Build Now** to execute the build job or commit the updated code to the repository, and the build will be executed automatically based on our configuration in **Build Triggers**.



It will check out all the latest available code in the source code repository against the local workspace on the machine where Jenkins is installed, as shown in the following figure.



Console Output

```

Started by user anonymous
[EnvInject] - Loading node environment variables.
Building in workspace /root/.jenkins/jobs/CounterApp/workspace
Checking out a fresh workspace because there's no workspace at /root/.jenkins
/jobs/CounterApp/workspace
Cleaning local Directory .
Checking out https://ms/svn/JenkinsTest/trunk/CounterWebApp at revision
'2015-05-01T10:37:28.604 -0700'
A      .classpath
A      .project
AU     CounterWebApp.war
A      target
A      src
A      src/main
A      src/main/java
A      src/main/java/com
A      src/main/java/com/tinyclouds
A      src/main/java/com/tinyclouds/controller
A      src/main/java/com/tinyclouds/controller/BaseController.java
A      src/main/resources
A      src/main/resources/logback.xml
A      src/main/webapp
A      src/main/webapp/WEB-INF
A      src/main/webapp/WEB-INF/pages
A      src/main/webapp/WEB-INF/pages/index.jsp
A      src/main/webapp/WEB-INF/mvc-dispatcher-servlet.xml

```

Once the checkout process is completed, the build file execution based on the goals will start, and the build execution will be successful if all dependencies and files required for the build execution are available in the local workspace, as shown in the following figure.

```
[INFO] Installing /root/.jenkins/jobs/CounterApp/workspace/target
/CounterWebApp.war to /root/.m2/repository/com/tinyclouds/CounterWebApp
/1.0-SNAPSHOT/CounterWebApp-1.0-SNAPSHOT.war
[INFO] Installing /root/.jenkins/jobs/CounterApp/workspace/pom.xml to /root/.m2
/repository/com/tinyclouds/CounterWebApp/1.0-SNAPSHOT/CounterWebApp-
1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.307 s
[INFO] Finished at: 2015-05-01T10:37:41-08:00
[INFO] Final Memory: 15M/36M
[INFO] -----
[JENKINS] Archiving /root/.jenkins/jobs/CounterApp/workspace/pom.xml to
com.tinyclouds/CounterWebApp/1.0-SNAPSHOT/CounterWebApp-1.0-SNAPSHOT.pom
[JENKINS] Archiving /root/.jenkins/jobs/CounterApp/workspace/target
/CounterWebApp.war to com.tinyclouds/CounterWebApp/1.0-SNAPSHOT/CounterWebApp-
1.0-SNAPSHOT.war
channel stopped
Deploying /root/.jenkins/jobs/CounterApp/workspace/CounterWebApp.war to
container Tomcat 7.x Remote
[/root/.jenkins/jobs/CounterApp/workspace/CounterWebApp.war] is not deployed.
Doing a fresh deployment.
Deploying [/root/.jenkins/jobs/CounterApp/workspace/CounterWebApp.war]
Deploying /root/.jenkins/jobs/CounterApp/workspace/target/CounterWebApp.war to
container Tomcat 7.x Remote
Redeploying [/root/.jenkins/jobs/CounterApp/workspace/target/CounterWebApp.war]
Undeploying [/root/.jenkins/jobs/CounterApp/workspace/target/CounterWebApp.war]
Deploying [/root/.jenkins/jobs/CounterApp/workspace/target/CounterWebApp.war]
Finished: SUCCESS
```

Build execution with test cases

Jenkins allows JUnit-format test results to be published on the dashboard. We need not install any specific plugin for this. If we have test cases already written in JUnit, then it is easy to execute them. Make sure to create a goal or task in the build file for test case execution. In Build Job configuration, click on **Post-build Actions** and select **Publish JUnit test result report**. Provide the location for the **Test report XMLs** files and save the build job configuration.

Publish JUnit test result report

Test report XMLs:

[Fileset 'includes' setting](#) that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is [the workspace root](#).

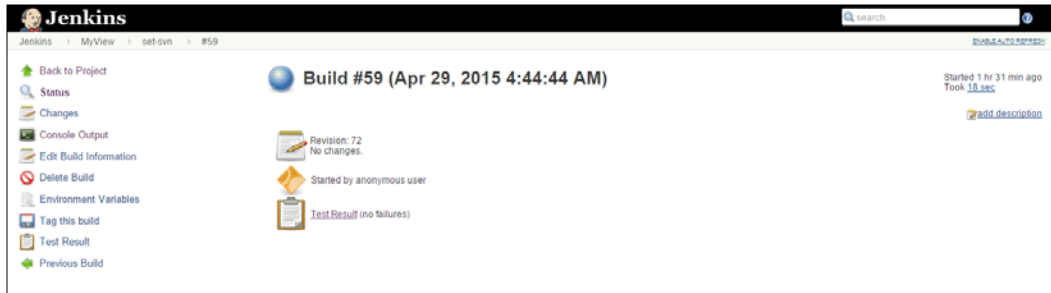
☐ Retain long standard output/error

Health report amplification factor:

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Delete

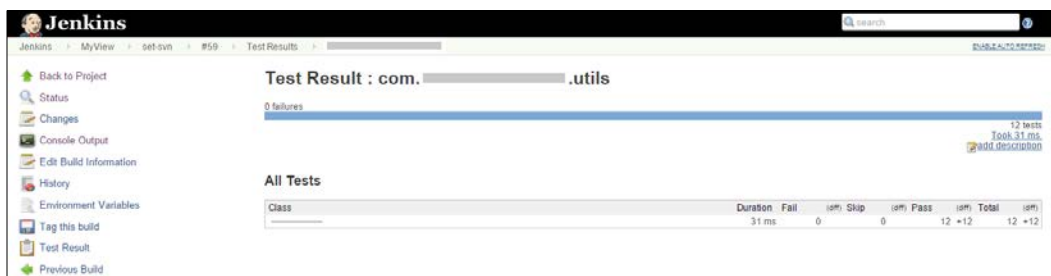
Execute the build by clicking on **Build Now**. Once the build has finished, click on the **Test Result** link on the dashboard.



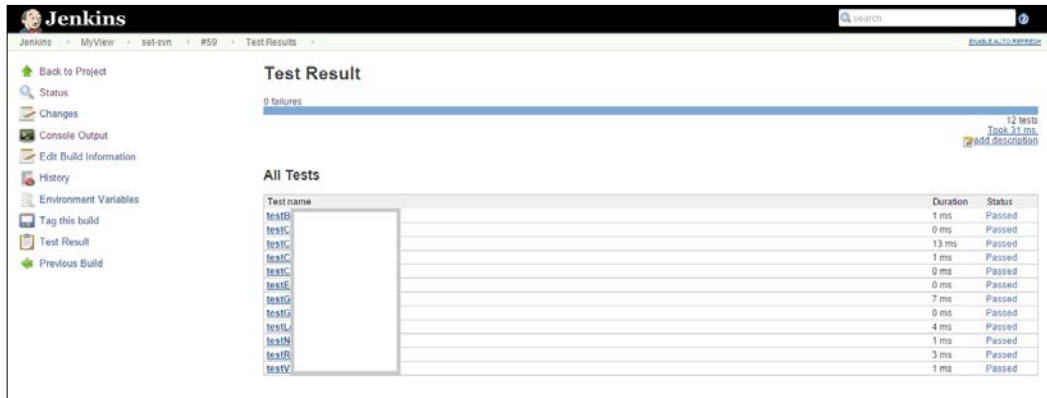
Click on the package link to get detailed test results on the summary page.



Click on the class link to get detailed test results on the page.



Verify all tests name, the duration, and the status, as shown in the following figure:



The screenshot shows the Jenkins 'Test Result' page. At the top, it says '0 failures' and '12 tests Took: 31 ms'. Below this is a table titled 'All Tests' with columns for 'Test name', 'Duration', and 'Status'. The table lists 12 tests, all of which are 'Passed'.

Test name	Duration	Status
testB	1 ms	Passed
testC	0 ms	Passed
testC	13 ms	Passed
testC	1 ms	Passed
testC	0 ms	Passed
testE	0 ms	Passed
testG	7 ms	Passed
testG	0 ms	Passed
testL	4 ms	Passed
testN	1 ms	Passed
testO	3 ms	Passed
testV	1 ms	Passed

Verify by clicking on the individual link of each test case on the Jenkins dashboard.

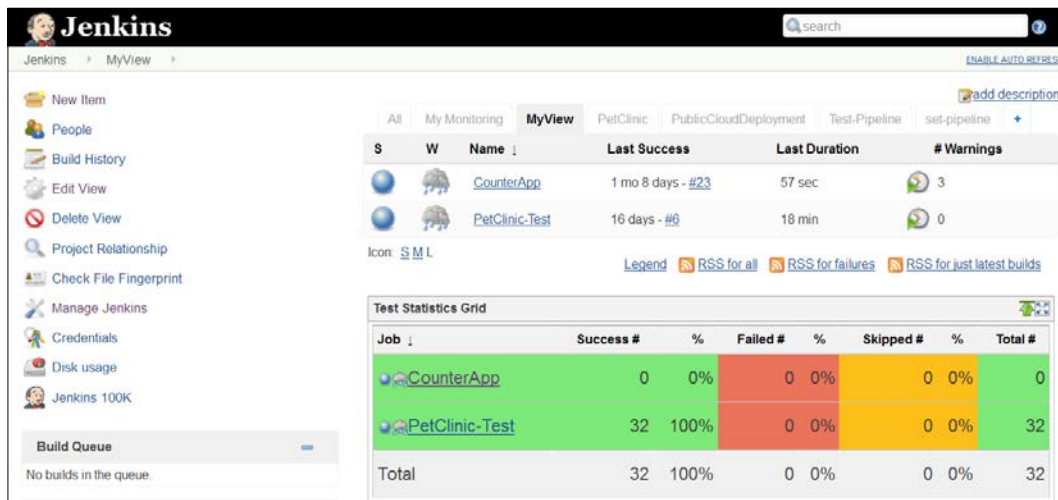


The screenshot shows the Jenkins 'Test Result' page for a specific test case. It displays 'Passed' in large blue text. Below this, it shows the test name 'com.swiftradius.HolidaysTest.testSetAllHolidaysInRange (from com. AllTests)'. At the bottom right, it says 'Took: 7 ms' and 'add description'.

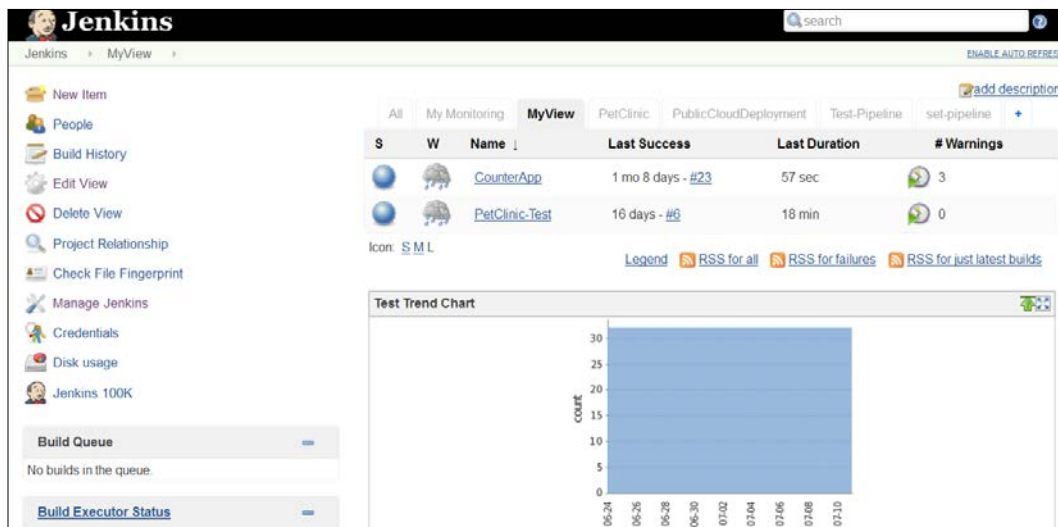
Test name	Duration	Status
com.swiftradius.HolidaysTest.testSetAllHolidaysInRange (from com. AllTests)	7 ms	Passed

We have already configured the Dashboard View plugin to display the Test Statistics Chart and the Test Trend Chart.

Verify the number of successful, failed or skipped tests, as well as the percentage on the customized view, as shown in the following screenshot.



Verify the Test Trend Chart on the Dashboard View.



Self-test questions

Q1. What is the objective of installing the Dashboard View plugin?

1. To have a portal-like view for Jenkins build jobs
2. To run test cases related to Jenkins build jobs
3. To display build results

Q2. Which are the fields available to create credentials for SVN?

1. **Scope, Username, Password, Description**
2. **Scope, Username, Password**
3. **Username, Password, Description**

Q3. What is the meaning of * * * * * in the **Schedule of Build Trigger** section?

1. Poll SCM Every Day
2. Poll SCM Every Hour
3. Poll SCM Every Minute
4. Poll SCM Every Second

Q4. What are the names of build files in Ant and Maven respectively?

1. pom.xml, build.xml
2. build.xml, pom.xml
3. pom.xml, root.xml
4. ant.xml, maven.xml

Summary

We are again at the part of the chapter that gives us a sense of achievement. In this chapter, we have covered how to customize the Jenkins dashboard and display test results based on the build job on the dashboard. We have also created our first build job for a sample Java application. We used build tools such as Ant and Maven for executing build and create artifacts. Finally, we have seen how test cases can be executed, and results can be displayed on the Jenkins portal.

In the next chapter, we will deploy the application to application server directly from Jenkins, and we will also cover an introduction to deploying applications on Amazon Web Services.

4

Implementing Automated Deployment

"Simplicity is prerequisite for reliability"

– Edsger Dijkstra

We have covered the concept of continuous integration, and we also know how to implement it using Jenkins. Now is the time to move to the next step in the application deployment pipeline, that is automated deployment. We will first understand the concept of continuous delivery and continuous deployment, before automated deployment into a Tomcat application server.

This chapter will take one step forward in the deployment pipeline by deploying artifacts in a local or remote application server. It will give an insight into the automated deployment and continuous delivery process.

- Overview of continuous delivery and continuous deployment
- Deploying a file from Jenkins to a Tomcat server

An overview of continuous delivery and continuous deployment

Continuous delivery is the extension of Continuous Integration practices. Application artifacts are production-ready in automated fashion but not deployed in production. Continuous deployment is the extension of continuous delivery, where changes in the application are finally deployed in production. Continuous delivery is a must for DevOps practices. Let's understand how to deploy application artifacts using Jenkins in the following sections.



For more details on continuous delivery and continuous deployment, visit:

<http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>

<http://martinfowler.com/books/continuousDelivery.html>

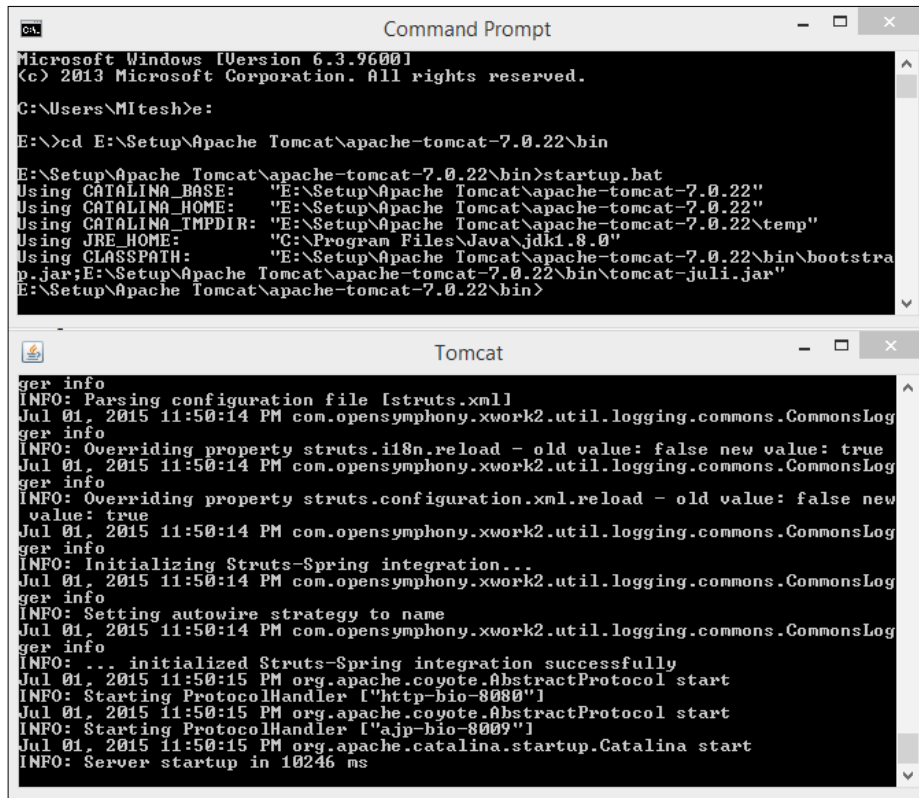
Installing Tomcat

Tomcat is an open source web server and servlet container developed by the **Apache Software Foundation (ASF)**. We will use Tomcat to deploy web applications.

1. Go to <https://tomcat.apache.org> and download Tomcat. Extract all the files to a relevant folder in your system.
2. Change the port number in `conf/server.xml` from 8080 to 9999.

```
<Connector port="9999" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

3. Open the terminal or Command Prompt based on your operating system. Go to the `tomcat` directory. Go to the `bin` folder, and run `startup.bat` or `startup.sh`. The following is an example of `startup.bat` on Windows.



The image shows two overlapping windows. The top window is a 'Command Prompt' titled 'C:\>'. It displays the following commands and output:

```

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Mitesh>E:
E:\>cd E:\Setup\Apache Tomcat\apache-tomcat-7.0.22\bin
E:\Setup\Apache Tomcat\apache-tomcat-7.0.22\bin>startup.bat
Using CATALINA_BASE:   "E:\Setup\Apache Tomcat\apache-tomcat-7.0.22"
Using CATALINA_HOME:   "E:\Setup\Apache Tomcat\apache-tomcat-7.0.22"
Using CATALINA_TMPDIR: "E:\Setup\Apache Tomcat\apache-tomcat-7.0.22\temp"
Using JRE_HOME:        "C:\Program Files\Java\jdk1.8.0"
Using CLASSPATH:       "E:\Setup\Apache Tomcat\apache-tomcat-7.0.22\bin\bootstrap.jar;E:\Setup\Apache Tomcat\apache-tomcat-7.0.22\bin\tomcat-juli.jar"
E:\Setup\Apache Tomcat\apache-tomcat-7.0.22\bin>

```

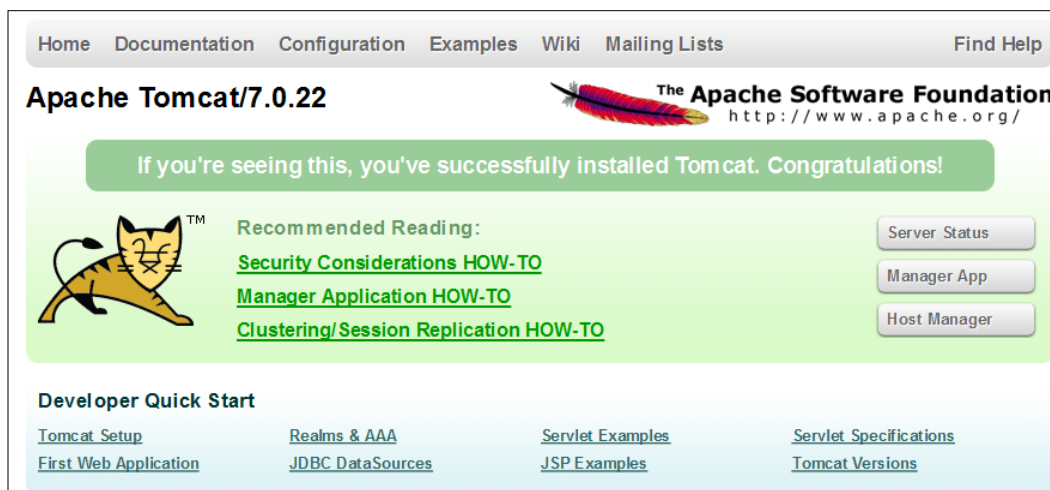
The bottom window is the 'Tomcat' console output, showing the following log messages:

```

ger info
INFO: Parsing configuration file [struts.xml]
Jul 01, 2015 11:50:14 PM com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: Overriding property struts.i18n.reload - old value: false new value: true
Jul 01, 2015 11:50:14 PM com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: Overriding property struts.configuration.xml.reload - old value: false new
value: true
Jul 01, 2015 11:50:14 PM com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: Initializing Struts-Spring integration...
Jul 01, 2015 11:50:14 PM com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: Setting autowire strategy to name
Jul 01, 2015 11:50:14 PM com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: ... initialized Struts-Spring integration successfully
Jul 01, 2015 11:50:15 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Jul 01, 2015 11:50:15 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Jul 01, 2015 11:50:15 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 10246 ms

```

4. Open your browser and visit <http://localhost:9999>. We can also access the Tomcat home page by using the IP address <http://<IP address>:9999>.



Deploying a war file from Jenkins to Tomcat

We will use the Deploy plugin available at <https://wiki.jenkins-ci.org/x/CAAjAQ> to deploy a war file into a specific container.

The Deploy plugin takes the war/ear file, and deploys it to a running local or remote application server at the end of a build.

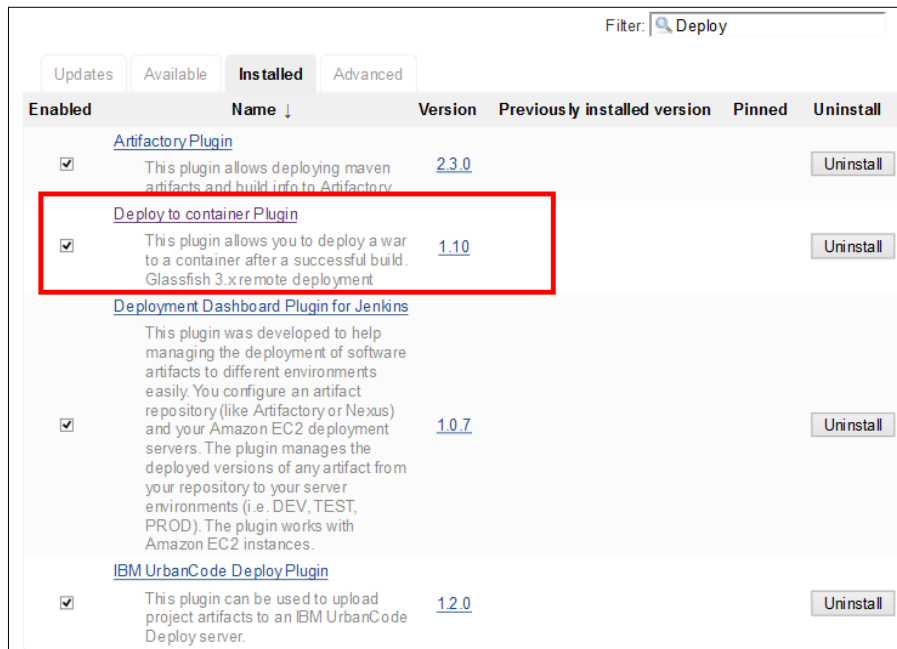
It supports the following containers:

- Tomcat: 4.x/5.x/6.x/7.x
- JBoss: 3.x/4.x
- Glassfish: 2.x/3.x

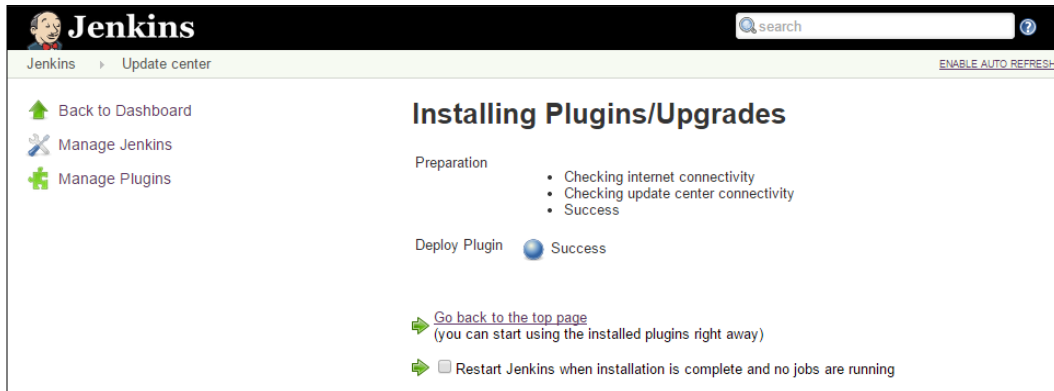
To deploy a war file in a Websphere container, use the Deploy WebSphere plugin available at <https://wiki.jenkins-ci.org/x/UgCkAg>.

To deploy a war file in a Weblogic container, use the WebLogic Deployer plugin available at <https://wiki.jenkins-ci.org/x/q4ahAw>.

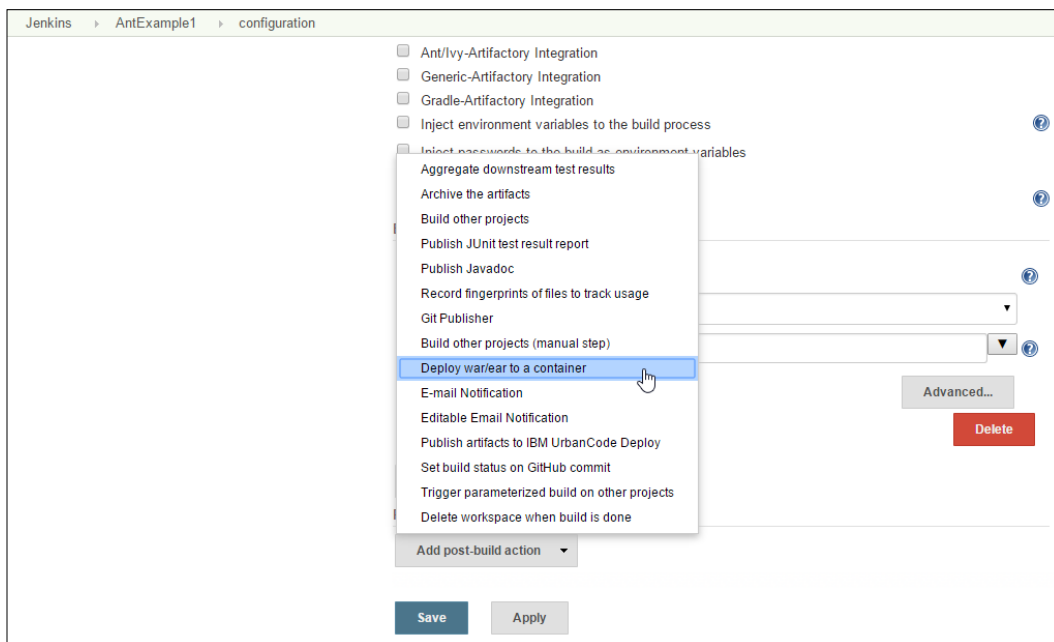
1. On the Jenkins dashboard, go to the **Manage Jenkins** link and then click on **Manage Plugins** and install **Deploy plugin**.



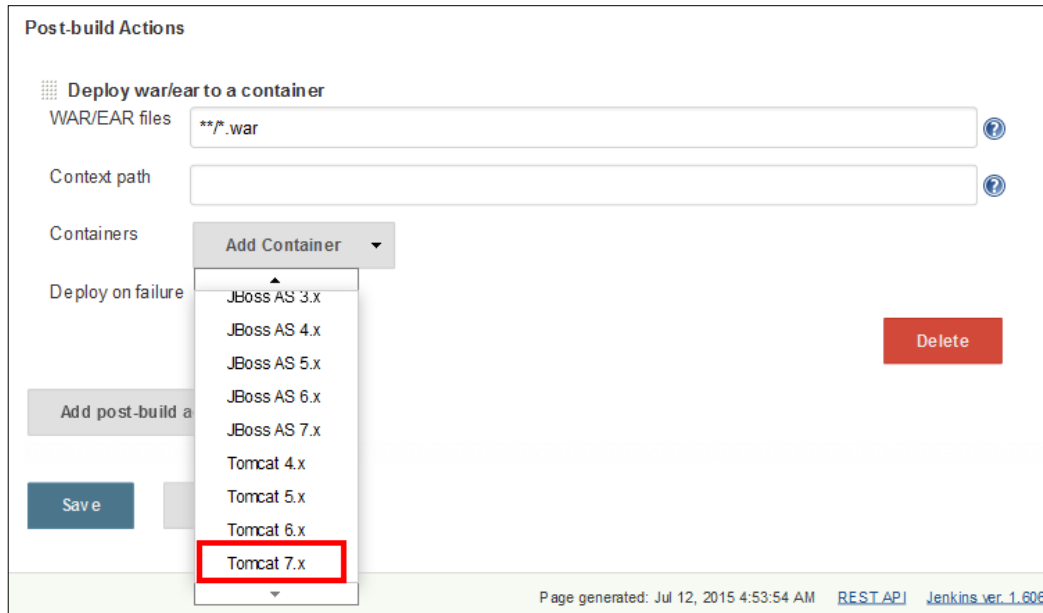
2. Wait until the installation of **Deploy Plugin** is complete.



3. Go to the Jenkins dashboard and select any build job. Click on the **Configure** link of the selected build job.
4. Click on the **Add post-build action** button on the configuration page of the relevant job and select **Deploy war/ear to container**, as shown in the following figure.



5. It will add **Deploy war/ear to a container** in the **Post-build Actions** section. Provide a **war** file path that is relative to the workspace, and select **Tomcat 7.x** as the container from the available list box, as shown in the following figure.



6. Provide **Manager user name** and **Manager password**; in tomcat-users.xml, and uncomment the following:

```
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
```

7. Add the following in the uncommented section:

```
<role rolename="manager-script"/>
<user username="mitesh51" password="*****" roles="manager-
script"/>
```

8. Restart Tomcat, visit <http://localhost:9999/manager/html>, and enter a username and password. Use the same username and password in Jenkins for Manager credentials.

Post-build Actions

Deploy war/ear to a container

WAR/EAR files ?

Context path ?

Containers

Tomcat 7.x

Manager user name

Manager password

Tomcat URL

Deploy on failure ☐

9. Click on **Build Now**.

Console Output

```

Started by user anonymous
[EnvInject] - Loading node environment variables.
Building on master in workspace /root/.jenkins/jobs/AntExample1/workspace
Checking out a fresh workspace because there's no workspace at /root/.jenkins
/jobs/AntExample1/workspace
Cleaning local Directory .
Checking out https://192.168.1.12/svn/MS/AntExample1 at revision
'2015-07-12T05:42:51.081 -0700'
A   license.txt
A   sonar-project.properties
A   src
A   src/com
A   src/com/[redacted]
A   src/com/[redacted] service
A   src/com/[redacted] service/[redacted].java
A   src/com/[redacted] service/[redacted].java
A   src/com/[redacted] domain
A   src/com/[redacted] domain/[redacted].java
A   src/com/[redacted] web
A   src/com/[redacted] web/[redacted].java
A   WebContent
A   WebContent/META-INF
A   WebContent/META-INF/MANIFEST.MF
A   WebContent/WEB-INF
A   WebContent/WEB-INF/lib
AU  WebContent/WEB-INF/lib/checkstyle-6.6.jar
AU  WebContent/WEB-INF/lib/org.springframework.beans-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/org.springframework.core-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/checkstyle-6.6-all.jar
AU  WebContent/WEB-INF/lib/org.springframework.web-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/commons-logging-1.0.4.jar
AU  WebContent/WEB-INF/lib/org.springframework.context.support-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/org.springframework.expression-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/antlr-runtime-3.0.jar
AU  WebContent/WEB-INF/lib/org.springframework.asm-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/org.springframework.web.servlet-3.0.0.M3.jar
AU  WebContent/WEB-INF/lib/org.springframework.context-3.0.0.M3.jar
A   WebContent/WEB-INF/dispatcher-servlet.xml
A   WebContent/WEB-INF/jsp
A   WebContent/WEB-INF/jsp/userSuccess.jsp
A   WebContent/WEB-INF/jsp/userForm.jsp
A   WebContent/WEB-INF/web.xml
A   WebContent/redirect.jsp
A   checkstyle_checks.xml
A   build.xml
At revision 26
no change for https://192.168.1.12/svn/MS/AntExample1 since the previous build

```

10. Once the build is complete, verify the console output of the deployment of the application in the Tomcat application server.

```
[workspace] $ /root/.jenkins/tools/hudson.tasks.Ant_AntInstallation/Ant1.9.4/bin/ant
Buildfile: /root/.jenkins/jobs/AntExample1/workspace/build.xml

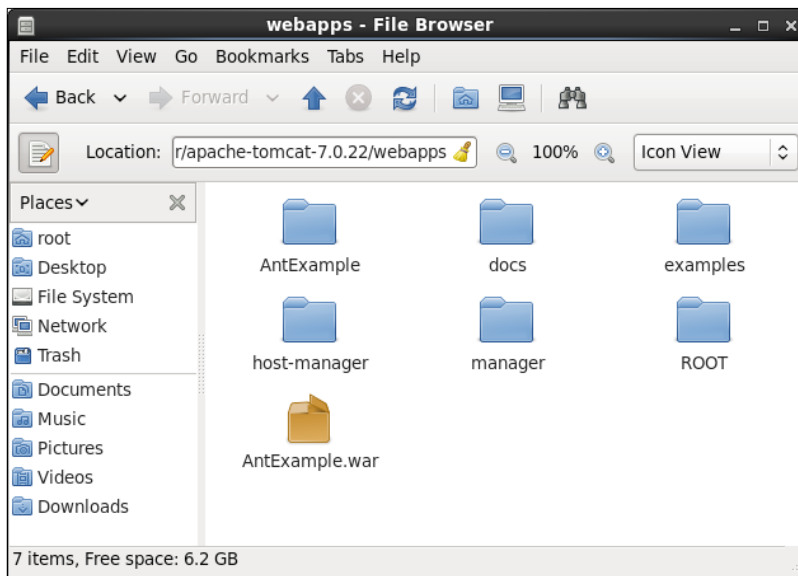
init:
[mkdir] Created dir: /root/.jenkins/jobs/AntExample1/workspace/build/classes
[mkdir] Created dir: /root/.jenkins/jobs/AntExample1/workspace/dist

compile:
[javac] /root/.jenkins/jobs/AntExample1/workspace/build.xml:16: warning:
'includantruntime' was not set, defaulting to build.sysclasspath=last; set to false
for repeatable builds
[javac] Compiling 4 source files to /root/.jenkins/jobs/AntExample1/workspace
/build/classes
[javac] Note: /root/.jenkins/jobs/AntExample1/workspace/src/com/vaannila
/web/UserController.java uses or overrides a deprecated API.
[javac] Note: Recompile with -Xlint:deprecation for details.

war:
[war] Building war: /root/.jenkins/jobs/AntExample1/workspace
/dist/AntExample.war

BUILD SUCCESSFUL
Total time: 13 seconds
Deploying /root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war to container
Tomcat 7.x Remote
  Redeploying [/root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war]
  Undeploying [/root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war]
  Deploying [/root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war]
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: SUCCESS
```

11. Verify the webapps directory in the Tomcat installation directory.



12. Verify the Tomcat manager, and check the status of an application in the Tomcat application server.




Tomcat Web Application Manager

Message: OK

Manager

[List Applications](#)
[HTML Manager Help](#)
[Manager Help](#)
[Server Status](#)

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/AntExample	None specified	AntExample	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

13. If the Tomcat server is installed on a remote server, then use the IP address in the Tomcat URL, as shown in the following figure:

Post-build Actions

Deploy war/ear to a container

WAR/EAR files

Context path

Containers

Tomcat 7.x

Manager user name

Manager password

Tomcat URL

Deploy on failure ☐

We only need to change the Tomcat URL in case of remote deployment.

Self-test questions

Q1. Continuous delivery and continuous deployment are the same.

1. True
2. False

Q2. How do you enable Tomcat manager access?

1. Start Tomcat
2. Modify `server.xml`
3. Modify `tomcat-users.xml`
4. Modify `web.xml`

Summary

Well done! We are at the end of the chapter; let's summarize what we have covered. We have understood the concept of continuous delivery and continuous deployment. The main concept we have covered here is the deployment of application artifacts in the specific application server after the build is successful.

In the next chapter, we will learn how to manage Jenkins on Cloud, and look at some case studies.

5

Hosted Jenkins

"Productivity is being able to do things that you were never able to do before"

–Franz Kafka

We have understood the concepts of continuous delivery and continuous deployment. We have also seen how to deploy the `war` file from Jenkins to the Tomcat server. Now, we will see how hosted Jenkins can be leveraged. Different service providers offer Jenkins as a service. We will see how OpenShift and CloudBees provide Jenkins to users.

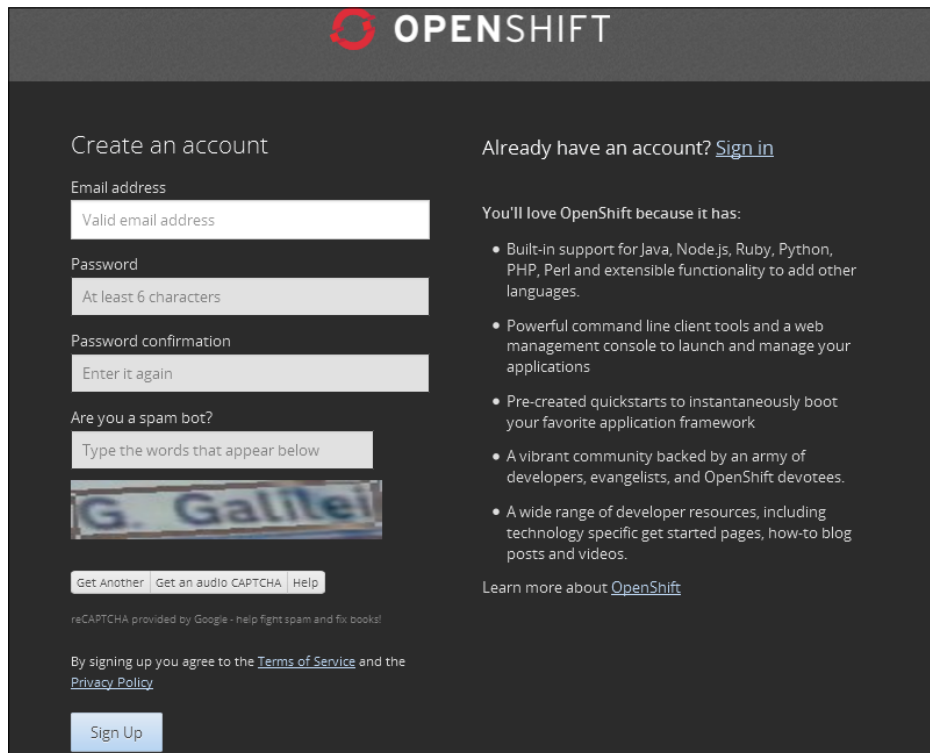
This chapter describes details on how to use hosted Jenkins, which is provided by popular PaaS providers, such as Red Hat OpenShift and CloudBees. This chapter also covers details on how various customers are using Jenkins based on their requirements. This chapter will explore details on how to use Cloud-related plugins in Jenkins for effective usage of Jenkins. We will cover the following topics in this chapter:

- Exploring Jenkins in OpenShift PaaS
- Exploring Jenkins in the Cloud – CloudBees
- An overview of CloudBees Enterprise Plugins
- Jenkins case studies from CloudBees

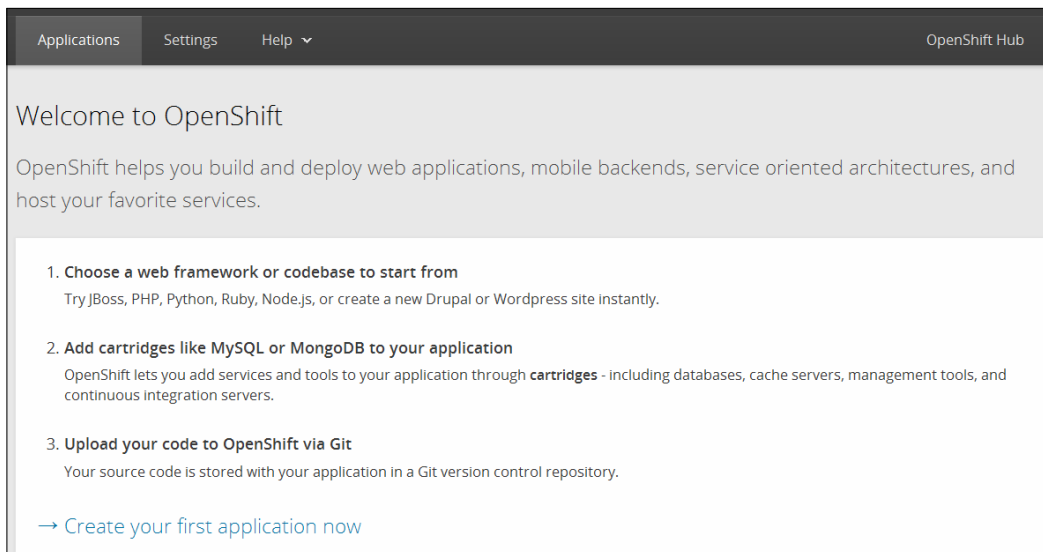
Exploring Jenkins in OpenShift PaaS

OpenShift Online is a public PaaS—application development and hosting platform from Red Hat. It automates the process of provisioning and deprovisioning, management, and scaling of applications. This supports command-line client tools and a web management console to launch and manage applications easily. The Jenkins app is provided by OpenShift Online. OpenShift Online has a free plan.

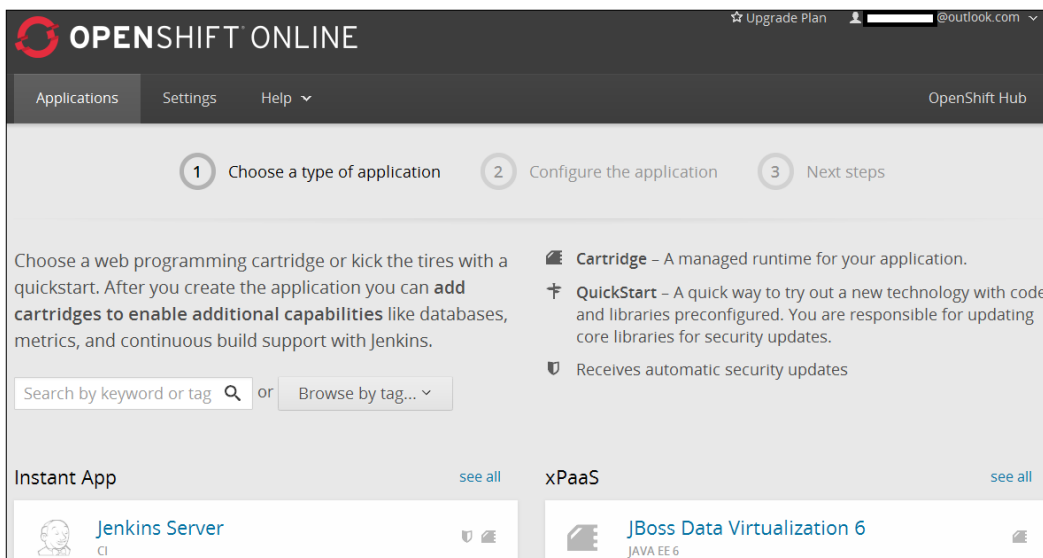
1. To sign up for OpenShift Online, visit <https://www.openshift.com/app/account/new>.

The image shows the OpenShift account creation page. At the top, the OpenShift logo is displayed. Below it, the page is divided into two main sections. The left section, titled 'Create an account', contains a form with the following fields: 'Email address' (with a placeholder 'Valid email address'), 'Password' (with a placeholder 'At least 6 characters'), 'Password confirmation' (with a placeholder 'Enter it again'), and a reCAPTCHA challenge showing the text 'G. Galilei'. Below the reCAPTCHA are links for 'Get Another', 'Get an audio CAPTCHA', and 'Help'. At the bottom of this section is a 'Sign Up' button. The right section, titled 'Already have an account? Sign in', lists several benefits of OpenShift: built-in support for various languages, powerful command-line and web management tools, pre-created quickstarts, a vibrant community, and a wide range of developer resources. At the bottom of this section is a link to 'Learn more about OpenShift'.

2. Once you sign up, you will get the welcome screen at <https://openshift.redhat.com/app/console/applications>.
3. Click on **Create your first application now**.



4. Choose a type of application, in our case, select **Jenkins Server**.



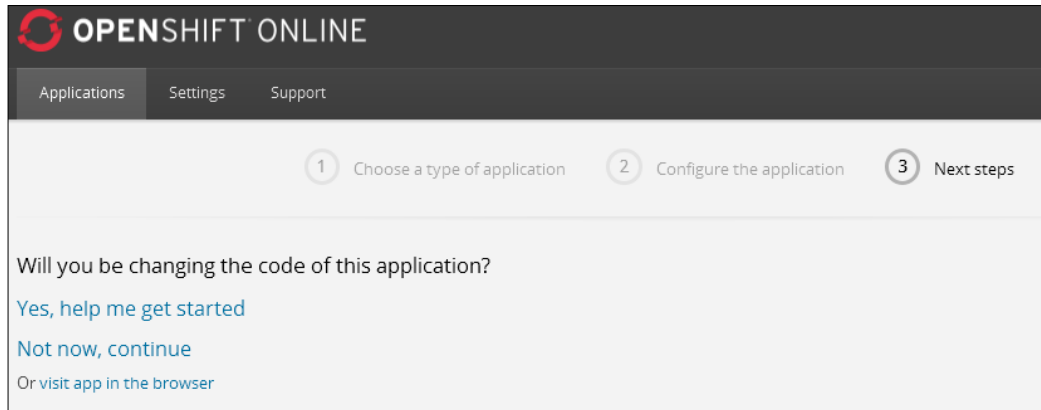
5. Give **Public URL** for your Jenkins server, as shown in the following screenshot:

The screenshot shows the 'Configure the application' step (step 2) in the OpenShift Hub interface. The top navigation bar includes 'Applications', 'Settings', and 'Help'. The main content area has a progress indicator with three steps: '1 Choose a type of application', '2 Configure the application' (current), and '3 Next steps'. Under 'Based On', the 'Jenkins Server Cartridge' is selected, with a description: 'Jenkins is a continuous integration (CI) build server that is deeply integrated into OpenShift. See the Jenkins info page for more. Based on Jenkins 1.509+'. Below this, there are links to 'http://www.jenkins-ci.org', a star icon for 'OpenShift maintained', and a shield icon for 'Receives automatic security updates'. The 'Public URL' section shows a text input field with 'http://', a highlighted input field containing 'jenkins', and a suffix '-msclouds.rhcloud.com'. A note states: 'OpenShift will automatically register this domain name for your application. You can add your own domain name later.' The 'Source Code' section has two input fields: 'Optional URL to a Git repository' and 'Branch/tag'.

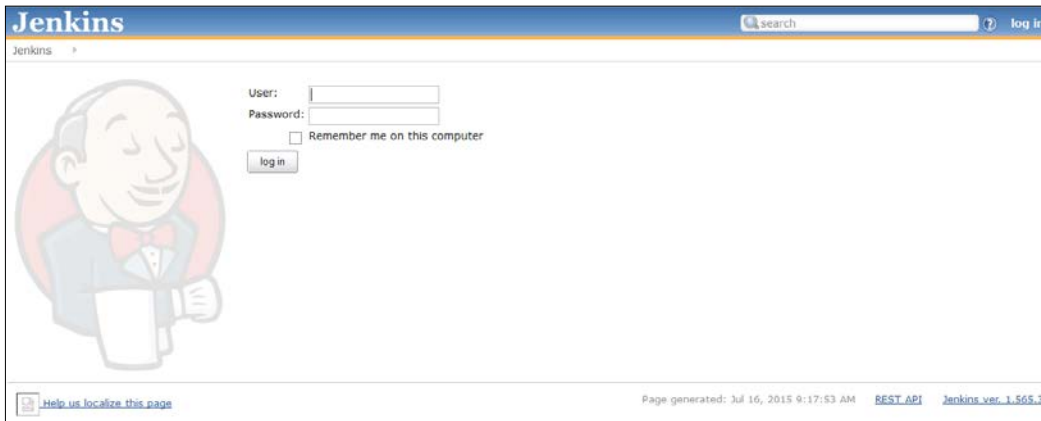
6. Click on **Create Application**.

The screenshot shows the 'Jenkins Server' configuration page. The 'Cartridges' section is set to 'Jenkins Server'. The 'Scaling' section is set to 'No scaling', with a note: 'OpenShift automatically routes web requests to your web gear. This application shares filesystem resources and can't be scaled.' The 'Region' section has several radio button options: 'No preference', 'aws-us-east-1' (selected), 'aws-eu-west-1', 'aws-ap-southeast-2', 'aws-us-west-1', and 'aws-eu-central-1'. Each region option has a warning message: 'WARNING: Small gears cannot be deployed to this region. Only production gears can be deployed to the EU Region (small, highcpu, medium, and large).', 'WARNING: This region is reserved for Dedicated Node Service', and 'WARNING: This region is reserved for Dedicated Node Service'. At the bottom, there are 'Back' and 'Create Application' buttons, and a '+1 @' icon.

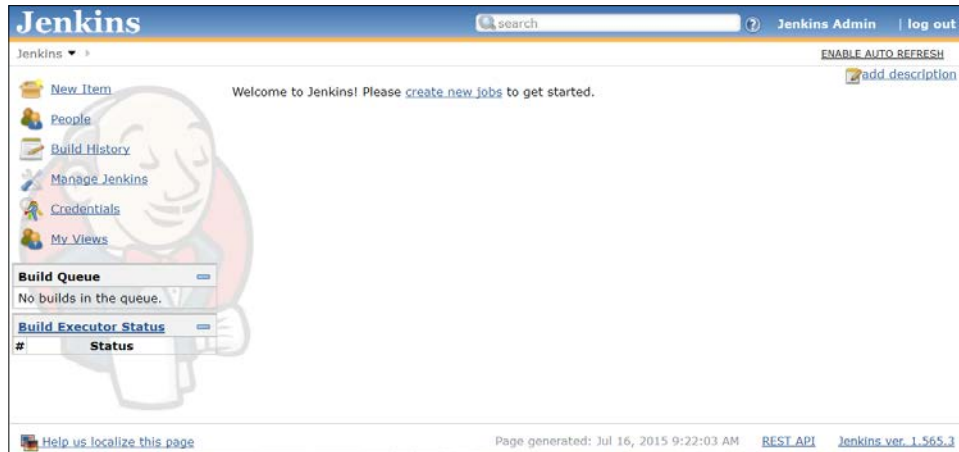
7. Click on **visit app in the browser**.



8. Access the Jenkins in the web browser. Then, log in with the provided credentials in the OpenShift dashboard.



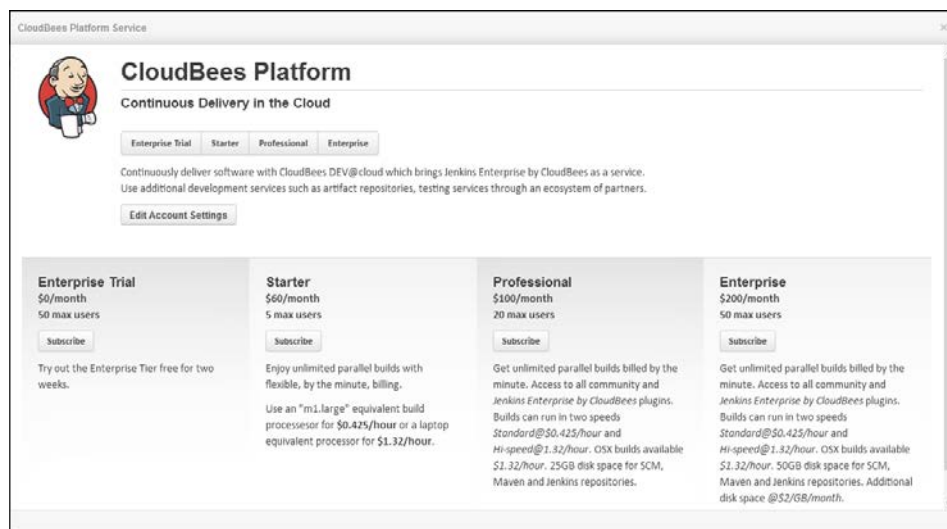
9. The following is the screenshot of the Jenkins dashboard:



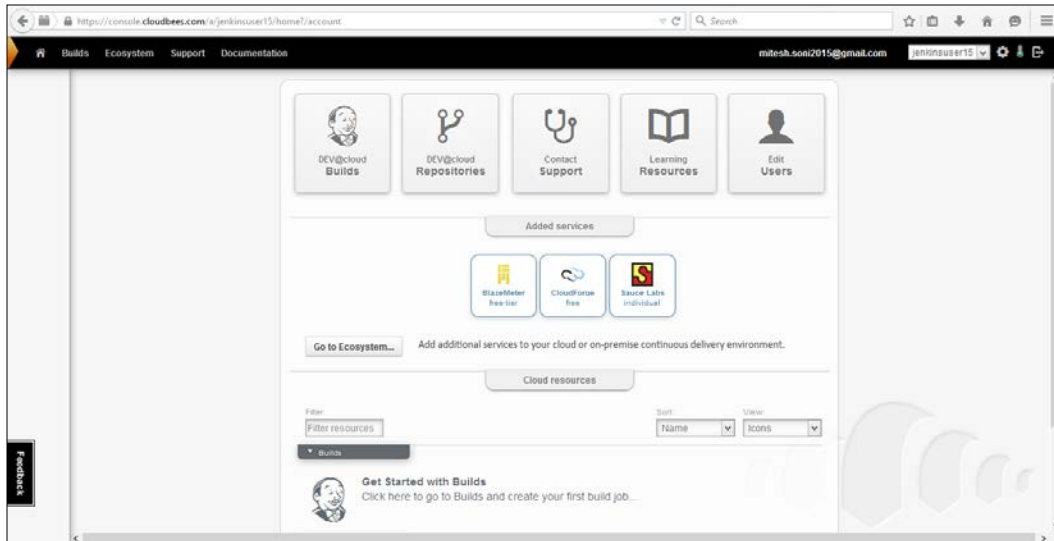
Exploring Jenkins in the Cloud – CloudBees

DEV@cloud is a hosted Jenkins service in a secure, multi-tenanted environment managed by CloudBees. It runs a specific version of Jenkins, along with a selected version of plugins which are well supported with that version. All updates and patches are managed by CloudBees, and limited customization is available.

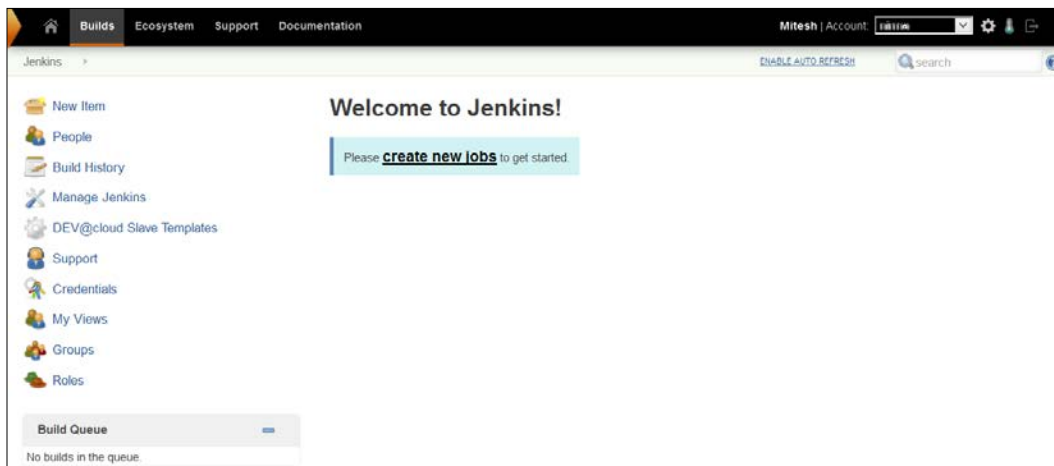
1. Go to <https://www.cloudbees.com/products/dev> and subscribe.



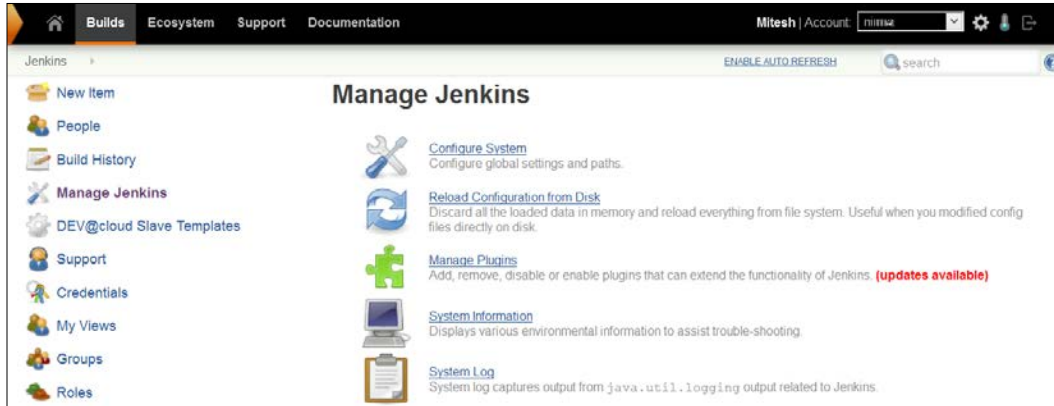
2. Once we complete subscription process, we will get the dashboard of CloudBees, as shown in following screenshot. Click on **Builds**.



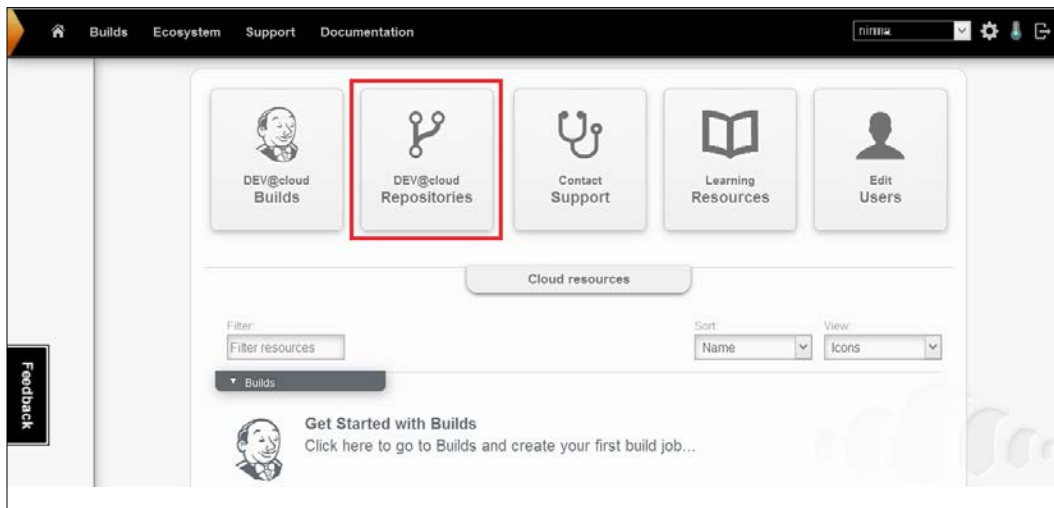
3. We will get the Jenkins dashboard, as shown in the following screenshot:



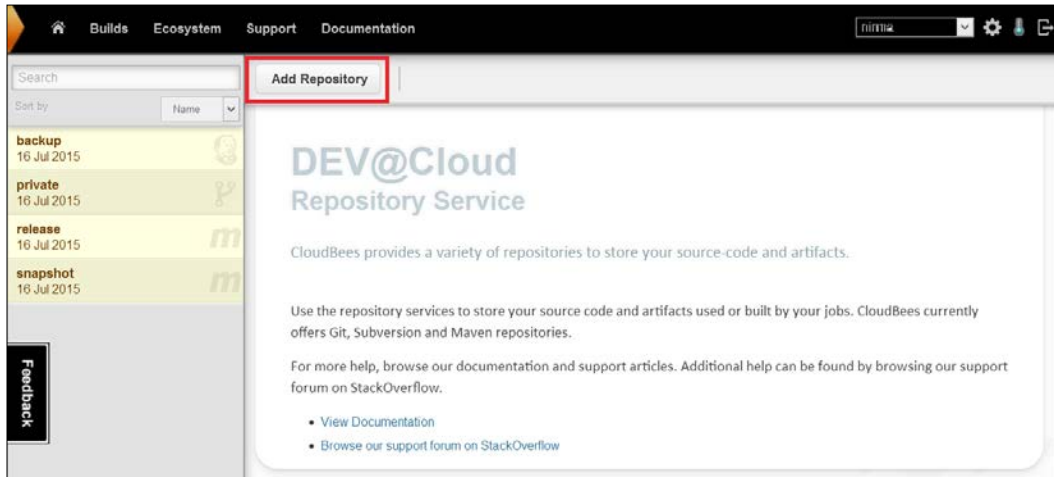
4. Click on **Manage Jenkins** to configure and install plugins.



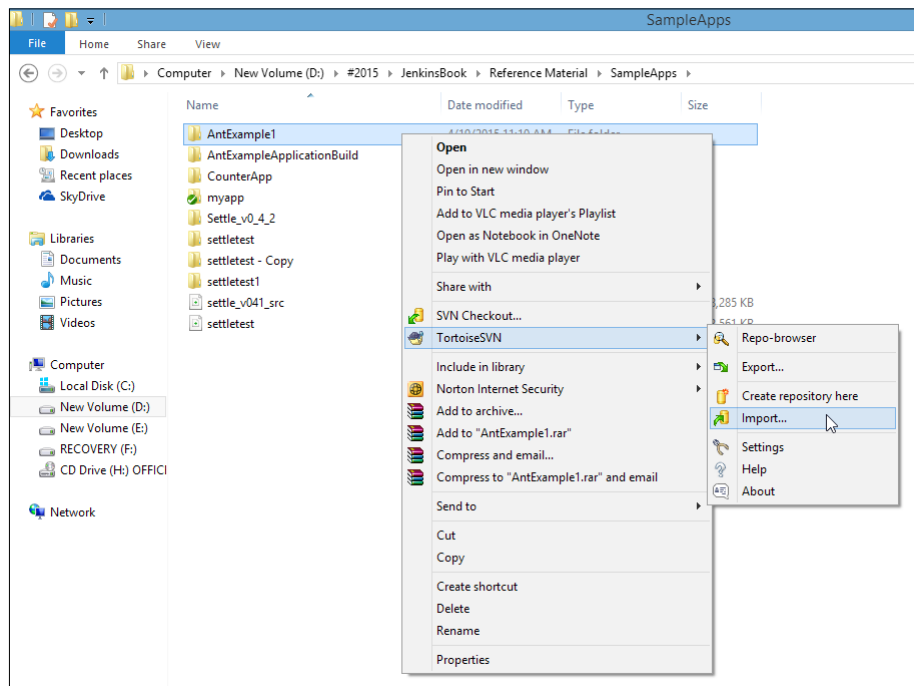
 Before configuring a build job, we need to store the source code of an application in the repository service provided by CloudBees. Click on **Ecosystem**, and then click on **Repositories**.



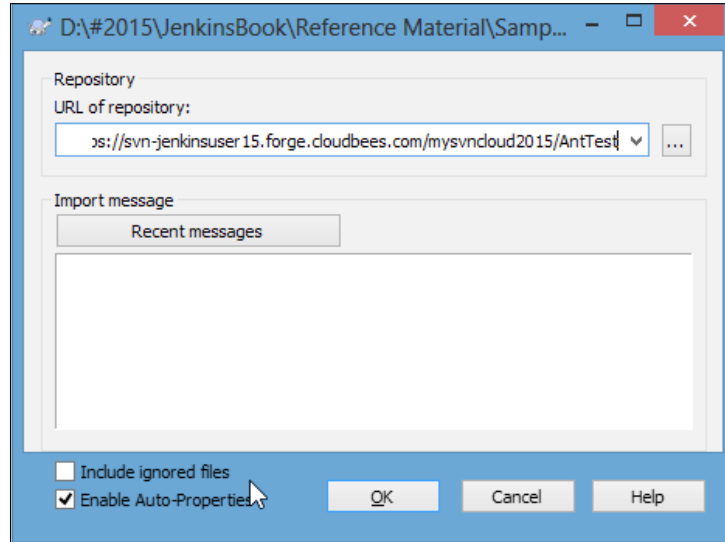
- Click on the subversion repositories or **Add Repository**, and get the URL of the repository.



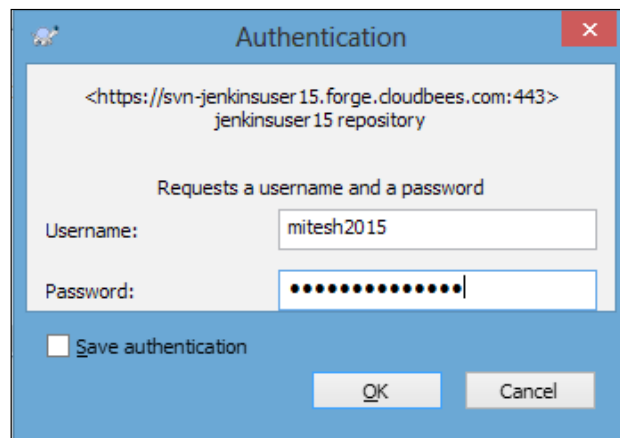
- Click on the application folder to import it into the subversion repository provided by CloudBees. Use TortoiseSVN or any SVN client to import the code.



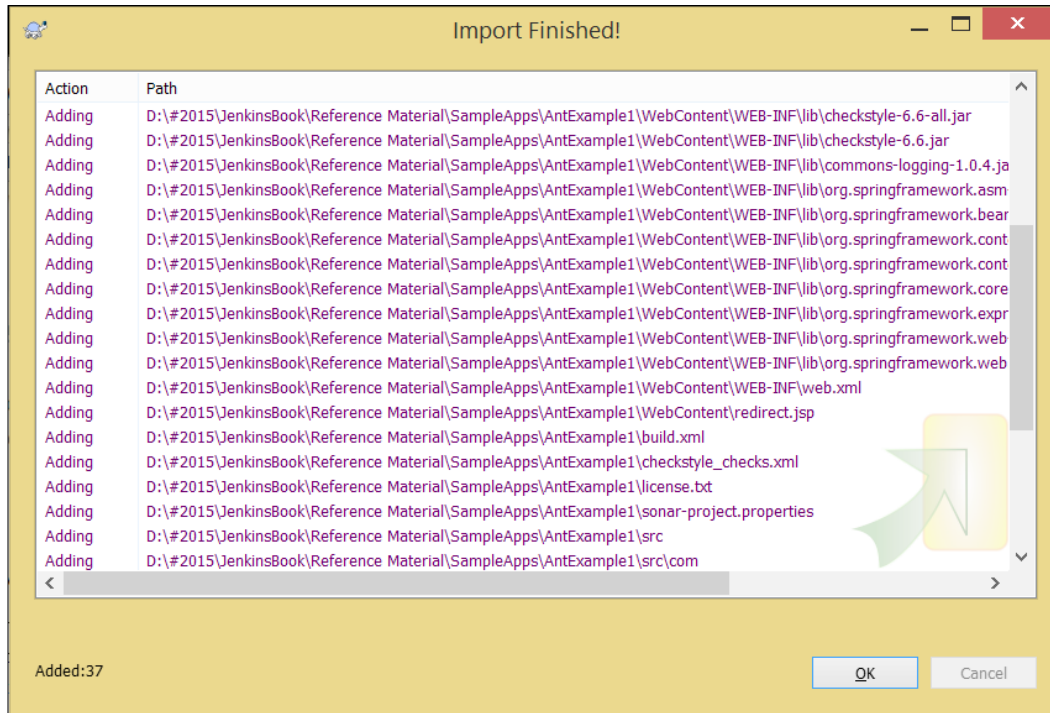
7. Provide the URL of a repository we copied from CloudBees, and click on **OK**.



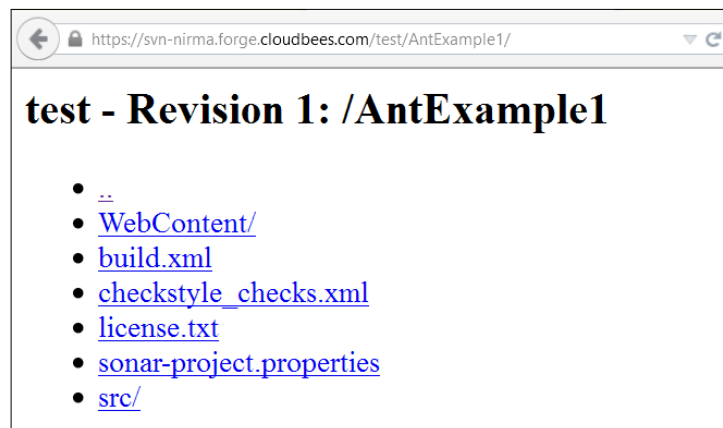
8. Provide authentication information (the username and password are same as our CloudBees account).
Click on **OK**.



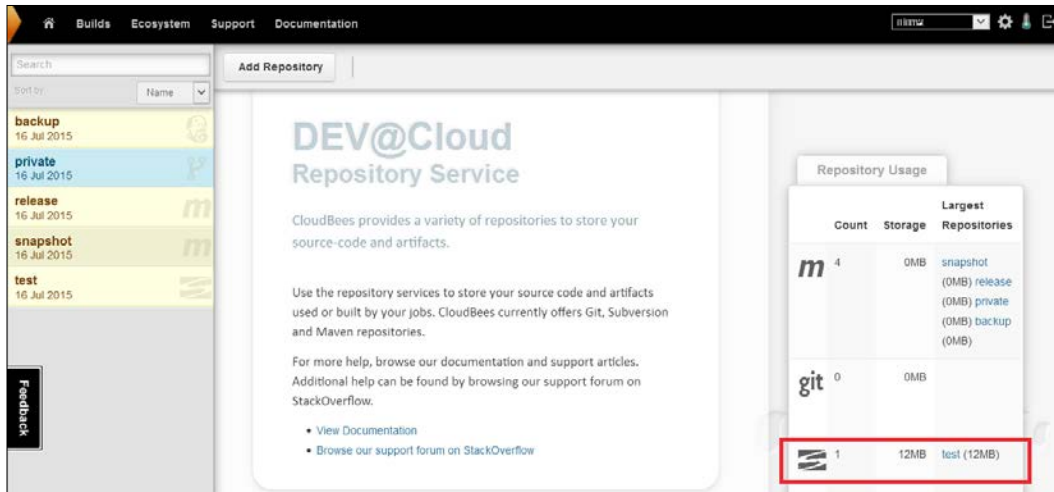
The import process will take some time based on the size of the source files.



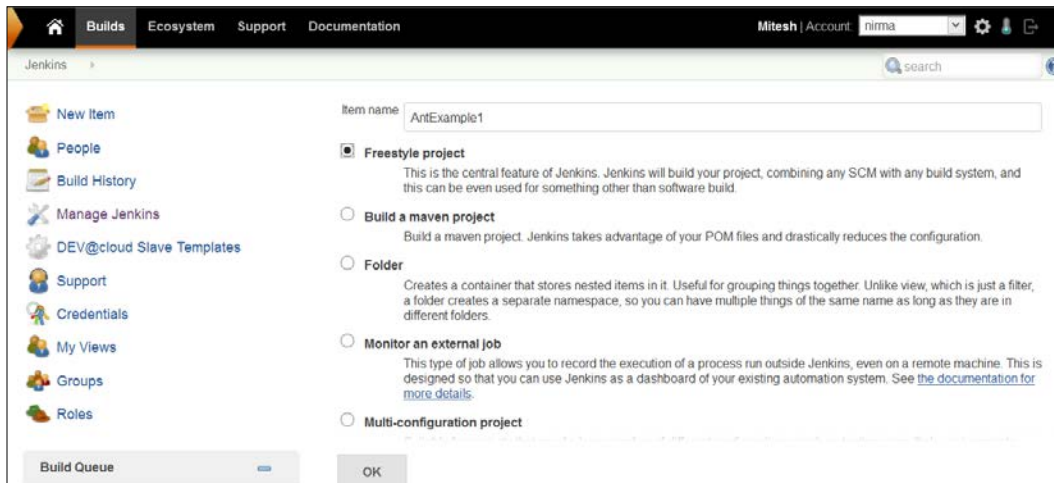
9. Verify the Repository URL on the browser, and we will find the recently imported project in it.



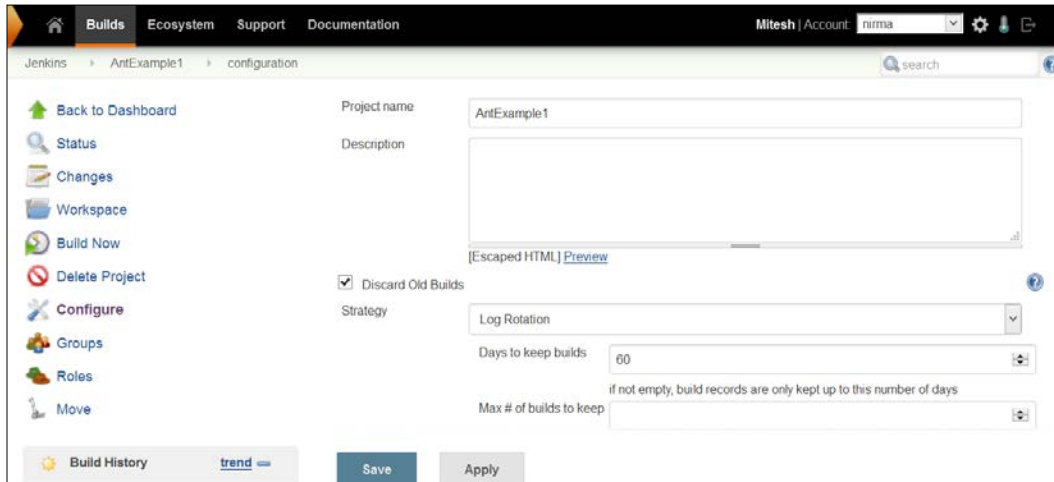
10. Verify the Jenkins dashboard after the successful import operation.



11. Click on **New Item** on the Jenkins dashboard. Select **Freestyle project**, and provide a name for a new build job. Click on **OK**.

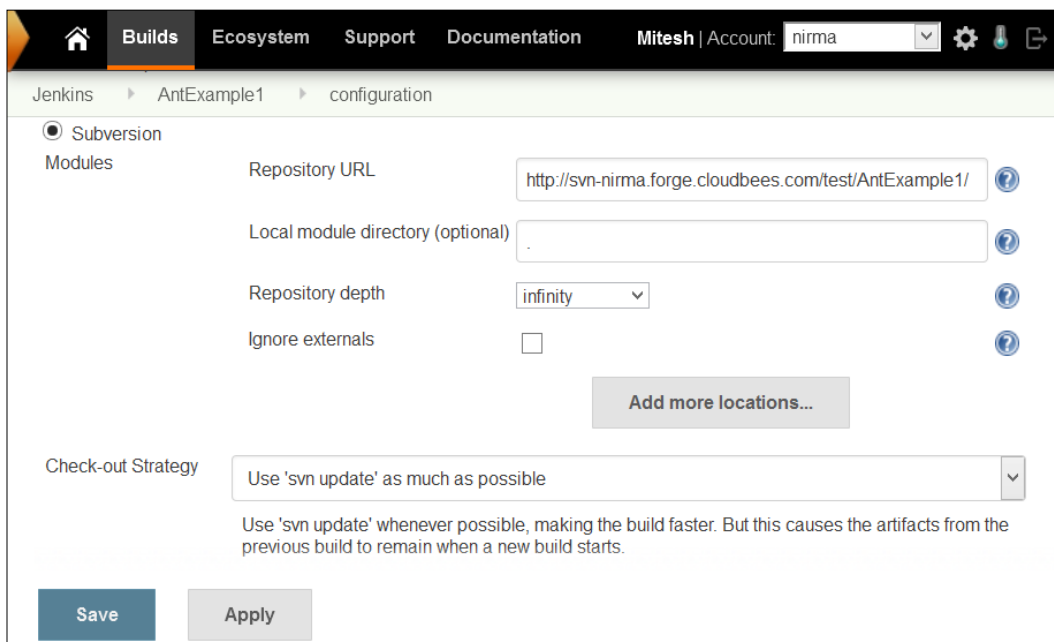


12. The configuration page will allow us to configure various settings specific to the build job.



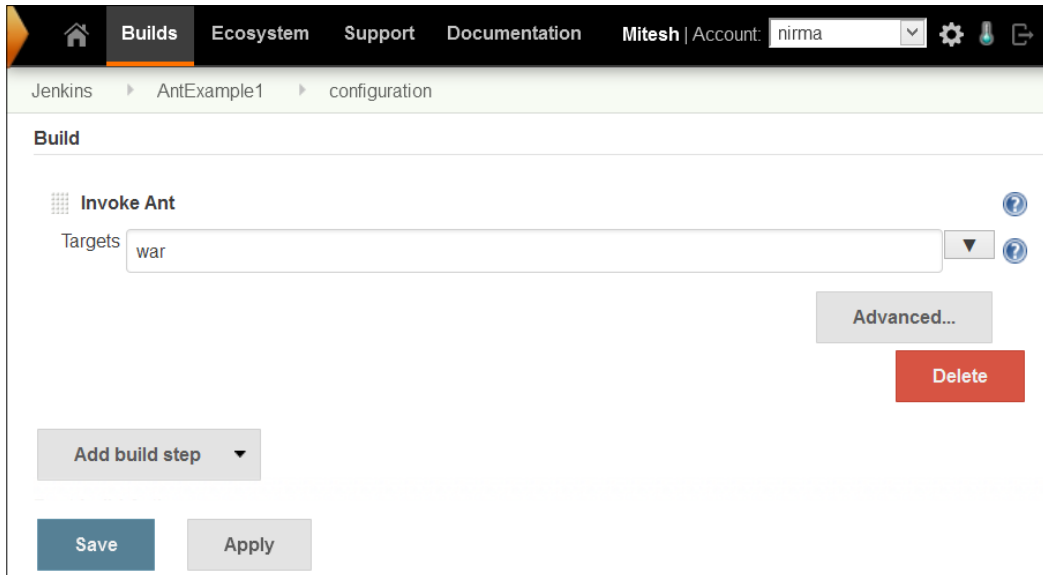
The screenshot shows the Jenkins configuration page for a job named 'AntExample1'. The left sidebar contains navigation links: Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure (selected), Groups, Roles, and Move. The main configuration area includes: Project name (AntExample1), Description (empty text area), Discard Old Builds (checked), Strategy (Log Rotation), Days to keep builds (60), and Max # of builds to keep (empty). A 'Build History' section at the bottom left shows a 'trend' link. 'Save' and 'Apply' buttons are at the bottom right.

13. Configure the **Subversion** repository in the build job.

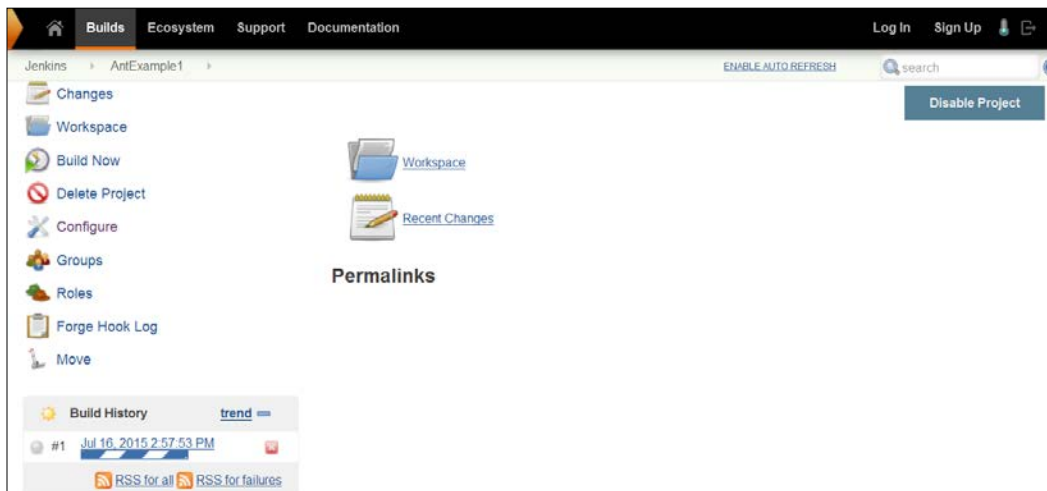


The screenshot shows the 'Subversion' configuration section of the Jenkins job. The 'Subversion' radio button is selected. Under 'Modules', there are fields for: Repository URL (http://svn-nirma.forge.cloudbees.com/test/AntExample1/), Local module directory (optional) (.), Repository depth (infinity), and Ignore externals (unchecked). An 'Add more locations...' button is below these fields. The 'Check-out Strategy' is set to 'Use 'svn update' as much as possible'. A descriptive note below states: 'Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.' 'Save' and 'Apply' buttons are at the bottom.

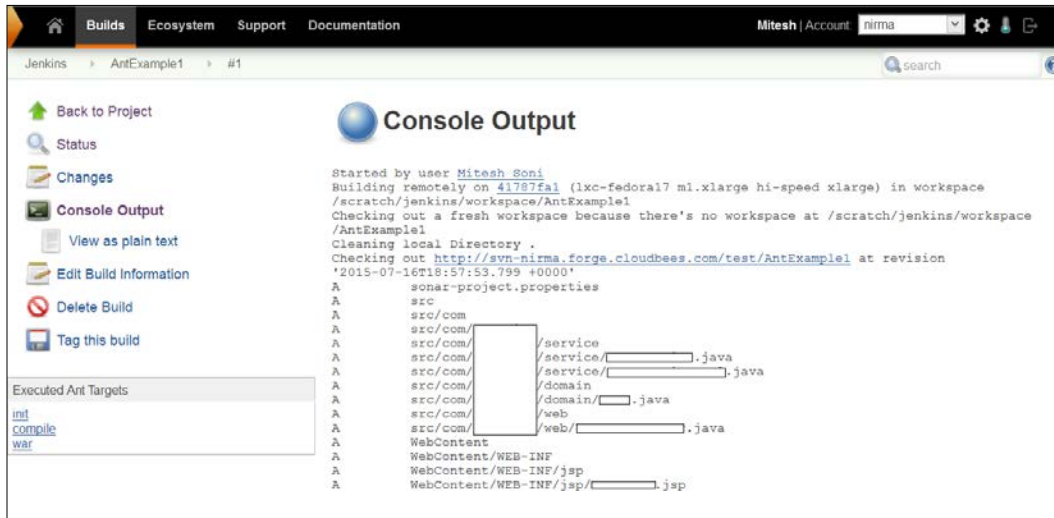
14. Click on **Apply**, and then click on **Save**.



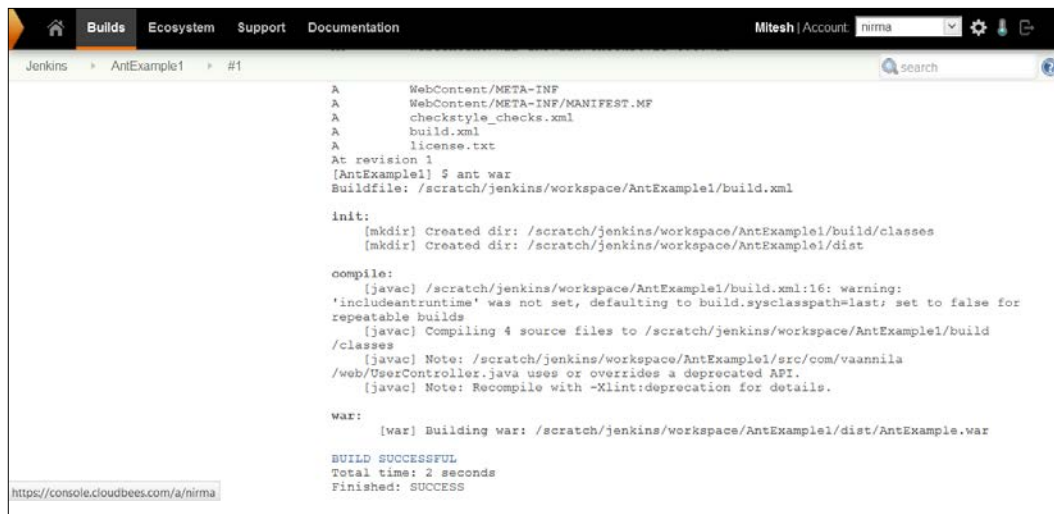
15. Click on **Build Now**.



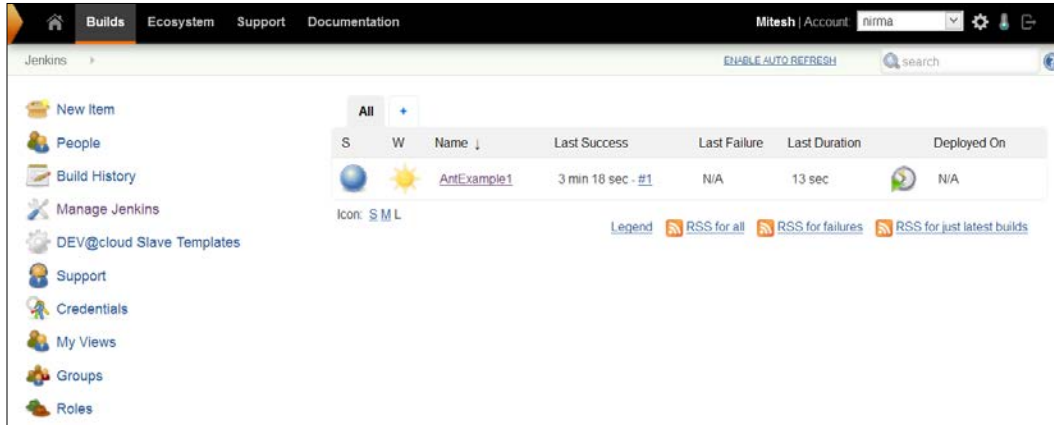
Verify Console Output.



Then, it will compile the source files, and create a war file based on the build.xml file, as this is an Ant-based project.



16. Verify the Jenkins dashboard for a successful build.



An overview of CloudBees Enterprise Plugins

The following are some important CloudBees Enterprise Plugins:

Workflow Plugin

It is a complex task to manage software delivery pipelines, and developer and operations teams need to manage complex jobs that can take days to complete. The Workflow plugin supports complex pipelines. The plugin uses Groovy DSL for workflows, and it also provides the facility to pause and restart jobs, to and from both master and slave failures.

To read more on this, visit <https://www.cloudbees.com/products/cloudbees-jenkins-platform/team-edition/features/workflow-plugin>.

Checkpoints Plugin

Let's consider a scenario where a long running build job fails almost at its end phase. This can hamper delivery schedules. The Checkpoints plugin provides the facility to restart workflows at checkpoints. Hence, it eliminates delays due to master and slave failures. In addition, it can help to survive Jenkins and infrastructure failures.

To read more on this, visit <https://www.cloudbees.com/products/jenkins-enterprise/plugins/checkpoints-plugin>.

Role-based Access Control Plugin

Authentication and authorization plays a significant role in the security aspect. The authorization strategy can help to control access to Jenkins jobs effectively. It is also essential to set permissions at the project level and visibility. The **Role-based Access Control (RBAC)** plugin provided by CloudBees provides the following features:

- To define various security roles
- To assign rules to groups
- To assign roles globally or at an object level
- To delegate management of groups for specific objects to users

To read more about the Role-based Access Control Plugin, visit <https://www.cloudbees.com/products/jenkins-enterprise/plugins/role-based-access-control-plugin>.

High Availability Plugin

The downtime of Jenkins master caused by software or hardware affects the entire product team. It is vital to bring Jenkins master up in quick time, and this will take many hours. The High Availability plugin eliminates downtime due to master failures, by keeping multiple masters as backups. A backup master automatically boots up when the failure of the master is detected. This plugin makes failure detection and recovery an automatic process and not manual.

To read more on this, visit <https://www.cloudbees.com/products/jenkins-enterprise/plugins/high-availability-plugin>.

VMware ESXi/vSphere Auto-Scaling Plugin

Let's consider a scenario where you need multiple slaves for Jenkins running in your existing infrastructure to utilize underutilized capacity of your virtualized infrastructure based on VMware. The VMware vCenter Auto-Scaling plugin allows you to create slave machines that are available in your VMware-based virtualized infrastructure. It is possible to configure pools of virtual machines that have identical and multiple VMs.

The following actions are allowed on VMs:

- Power on
- Power off/suspend
- Revert to the last snapshot

To read more, visit <https://www.cloudbees.com/products/jenkins-enterprise/plugins/vmware-esxisphere-auto-scaling-plugin>.

To find details on all plugins provided by CloudBees, visit <https://www.cloudbees.com/products/jenkins-enterprise/plugins>.

Jenkins case studies from CloudBees

We will cover some case studies from CloudBees, where Jenkins is used effectively.

Apache jclouds

Apache jclouds is an open source multi-cloud toolkit that provides the facility to manage workloads on multiple clouds. It was created on the Java platform, and provides complete control to use cloud platform-specific features to create and manage applications. It provides seamless portability across various cloud platforms. Apache jclouds support 30 cloud providers and cloud software stacks such as Joyent, Docker, SoftLayer, Amazon EC2, OpenStack, Rackspace, GoGrid, Azure, and Google. Apache jclouds has a remarkable user base such as CloudBees, Jenkins, Cloudify, cloudsoft, Twitter, Cloudswitch, enStratus, and so on.

Challenge

The jclouds community uses Jenkins CI for continuous integration. Day by day, it was getting more difficult to manage and maintain Jenkins, and it was a costly affair. Managing Jenkins was a time-consuming and tedious task. Most of the time developers were involved in the managing of Jenkins, and not in writing the code to make jclouds more effective.

Solution

The jclouds team explored PaaS offerings available in the market and considered CloudBees, which will help them to eliminate infrastructure management and maintenance. It was recognized by the jclouds team that it is easy to shift the Jenkins CI work to DEV@cloud and immediately gain productivity benefits from developers. Almost 4 hours were saved weekly from the maintenance activity of Jenkins.

Benefits

- 100% focus on software development, by eliminating activities such as server reboots, server sizing, software updates, and patches, as they are automatically performed from within the CloudBees service
- 33% increase in developer productivity
- Technical support from CloudBees for Jenkins CI issues

To read more about this case study, visit <https://www.cloudbees.com/casestudy/jclouds>.

Global Bank

Global Bank is one of the top Global Financial Institutions. It offers corporate and investment banking services, private banking services, credit card services and investment management. It has a substantial international presence.

Challenge

Global Bank's existing process was suffering from a fragmented build process, non-approved software versions, and a lack of technical support. There was a pool of central control or management, and standardization of the process. Build assets were not accessible all the time. There was a need for secure automated process for application build services with audit capabilities. Jenkins provided standardization along with other benefits of a centralized management with robustness and the availability of useful plugins. After using open source Jenkins, the financial institution faced other challenges that were not available in open source Jenkins. More features were needed for approvals, security, backup, and audit.

Solution

To overcome existing challenges, Global Bank evaluated and selected CloudBees Jenkins Enterprise, considering the additional plugins for high availability, backup, security, and job organization, and the ability to obtain technical support for open source Jenkins and open source Jenkins plugins. Global Bank utilized technical support from CloudBees for setting up CloudBees Jenkins Enterprise.

Benefits

- RBAC Plugin provides security and additional enterprise-level functionality. The Folders plugin offers version control and ensures that only approved software versions are shared.
- Half a day of development time is saved per application, by eliminating the need of monitoring the local instance of the build for each application.
- Availability of technical support capabilities.

To read more, visit <https://www.cloudbees.com/casestudy/global-bank>.

Service-Flow

Service-Flow provides online integration services, to connect the disparate IT service management tools used by organizations and various stakeholders. It provides features to create ticket automatically, ticket information exchange, and ticket routing. It has adapters for many ITSM tools such as ServiceNow and BMC, as well as Microsoft Service Manager Fujitsu, Atos, Efecte, and Tieto.

Challenge

Service-Flow wanted to build its own service without using any of the generic integration tools for achieving agility. Service-Flow had several requirements, such as focus on agility, which required a platform for rapid development and frequent incremental updates, support for Jenkins, control over data, reliability, and availability.

Solution

Service-Flow used the CloudBees platform to build and deploy its ITSM integration service. DEV@cloud has been utilized by establishing the version control repository, coding first Java classes, setting up some basic Jenkins jobs, running unit tests, executing integration tests, and other quality checks. The Service-Flow service is in the cloud with a rapidly growing customer base by adding new features using the CloudBees platform.

Benefits

- Development time reduced by 50 percent with production release in three months
- Updates deployed multiple times a week without service downtime
- Availability of 99.999 percent achieved in production

To read more, visit <https://www.cloudbees.com/casestudy/service-flow>.

For more case studies, visit <https://www.cloudbees.com/customers>.

Self-test questions

Q1. What is true about Workflow Plugin provided by CloudBees?

1. To pause and restart jobs, to and from both master and slave failures
2. To manage software delivery pipelines
3. It uses Groovy DSL for workflows
4. All of the above

Q2. What are the features of RBAC Plugin provided by CloudBees?

1. To define various security roles
2. To assign rules to groups
3. To assign role globally or at an object level
4. All of the above

Q3. What actions can be performed by VMware ESXi/vSphere Auto-Scaling Plugin provided by CloudBees?

1. Power on
2. Power off/suspend
3. Revert to the last snapshot
4. All of the above

Summary

The interesting thing about the ending of a chapter is: each chapter that is ending leads you to a new beginning. We know how to configure, manage, and use Jenkins on Cloud service models such as PaaS, RedHat OpenShift, and CloudBees. We also covered some interesting enterprise plugins from CloudBees, which add a lot of flexibility and value. In the last section, we have all provided details on various case studies on how Jenkins proved to be beneficial to a lot of organizations, and how they leveraged functionality of Jenkins to gain a competitive edge.

6

Managing Code Quality and Notifications

"Limit your burden by making very small incremental changes"

–Anonymous

We saw how various customers are using Jenkins on Cloud, based on their requirements. We also saw cloud-based offerings from Red Hat OpenShift and CloudBees, and case studies to understand how Jenkins is used effectively. Now, it is time to know about additional aspects of code quality inspection and notification on build failure.

This chapter will teach you how to integrate static code analysis behavior into Jenkins. Code quality is an extremely vital feature that impacts application's effectiveness and by integrating it with sonar, Checkstyle, FindBugs, and other tools, the user gets an insight into problematic portions of code.

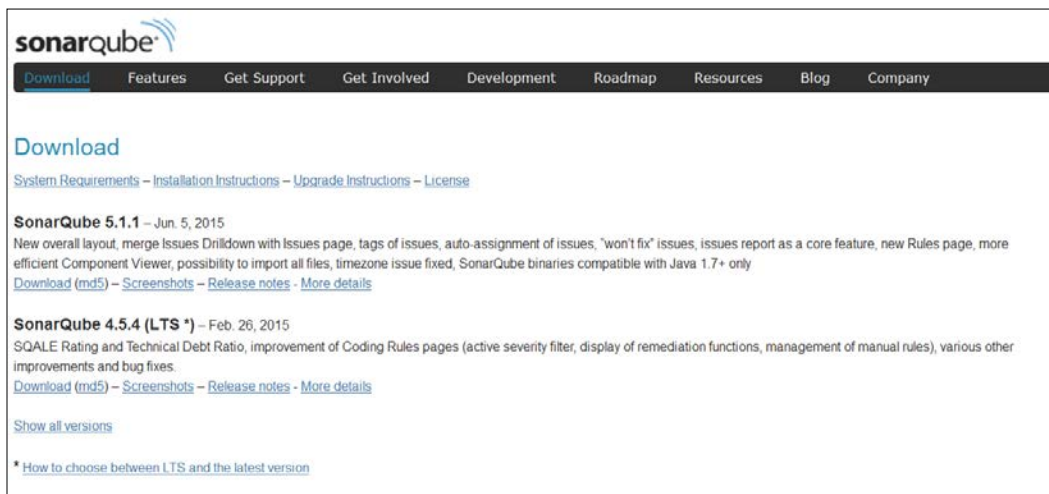
- Integration with Sonar
- Exploring Static code analysis Plugins
- E-mail Notifications on Build status

Integration with Sonar

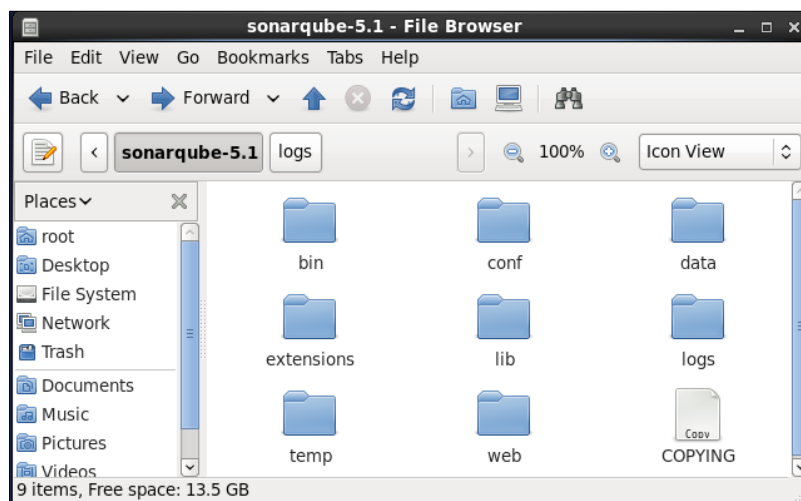
Quality of code is one of the important facets of DevOps culture. It provides quality checks that highlight the level of reliability, security, efficiency, portability, manageability, and so on. It helps to find bugs or possibility of bugs in the source code and sets culture to align with coding standards in the organization.

SonarQube is the open source platform for continuous inspection of code quality. It supports Java, C#, PHP, Python, C/C++, Flex, Groovy, JavaScript, PL/SQL, COBOL, Objective-C, Android development, and so on. It provides reports on coding standards, code coverage, complex code, unit tests, duplicated code, potential bugs, comments, design and architecture.

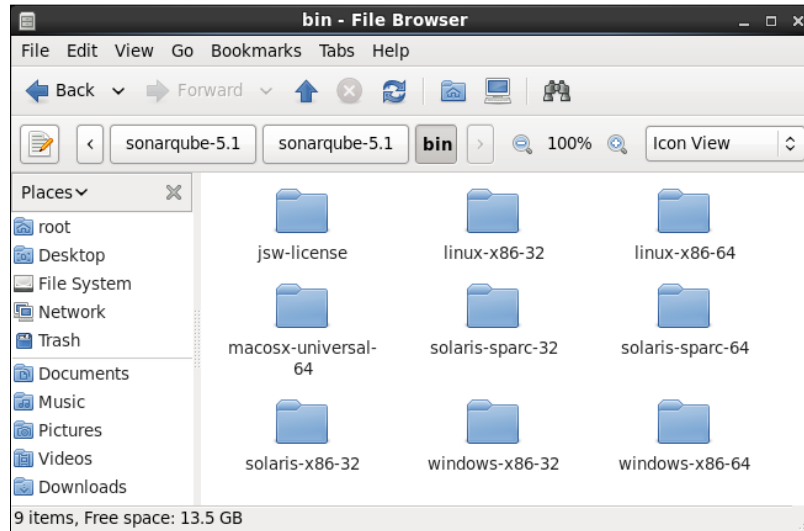
1. Go to <http://www.sonarqube.org/downloads/>, and download SonarQube 5.1.



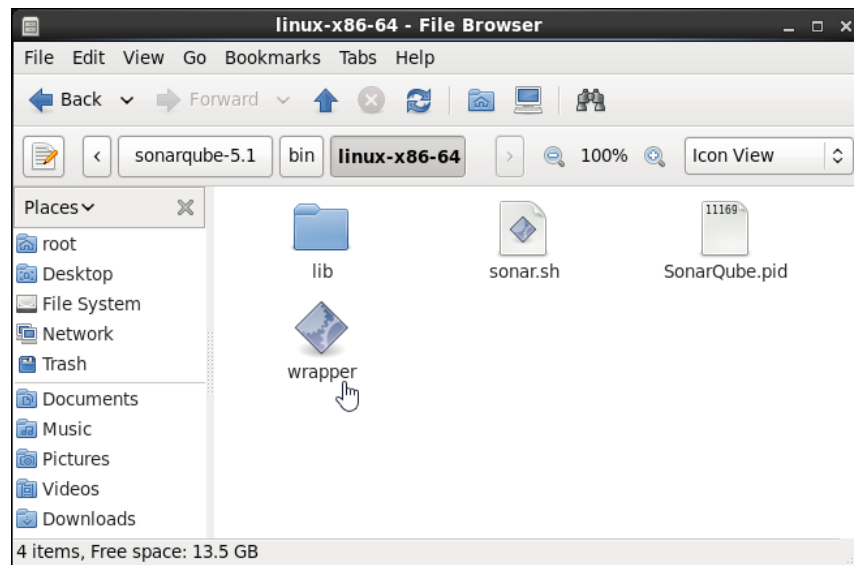
2. Extract files, and it will look similar to the following screenshot:



3. Go to the `bin` folder to run SonarQube based on the operating system on which you want to run Sonar.



4. Select a folder based on your platform, in our case, we are installing it on CentOS, and so we will select `linux-x86-64`.



- Open the terminal and go to the SonarQube home directory; go to `bin/linux-x86-64/` and run `sonar.sh`. We need to use parameters with `sonar.sh`, as shown in the following usage:

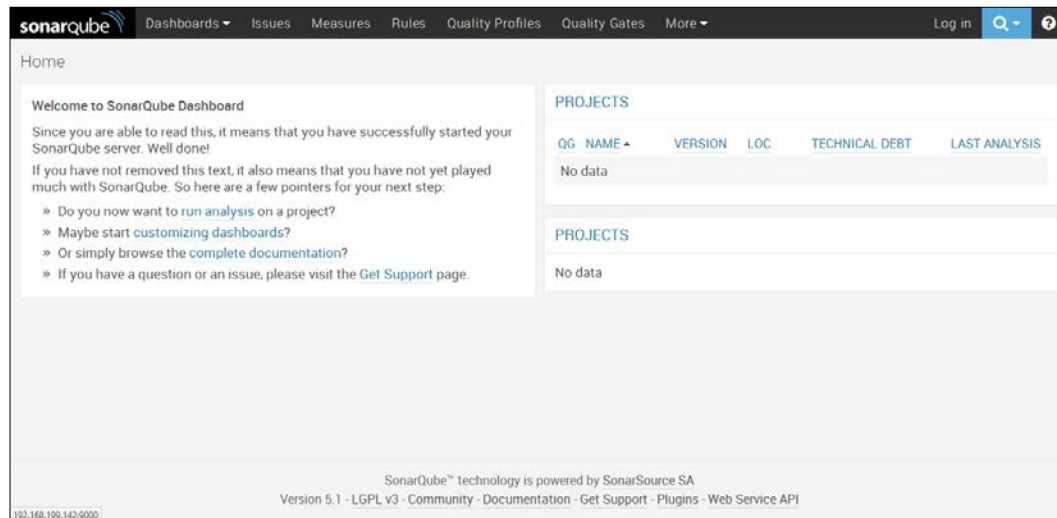
```
[root@localhost linux-x86-64]# ./sonar.sh

Usage: ./sonar.sh { console | start | stop | restart | status |
dump }
```

```
root@localhost:tmp/sonarqube-5.1/sonarqube-5.1/bin/linux-x86-64
File Edit View Search Terminal Tabs Help

root@localhost:tmp
root@localhost:tmp/sonarqube-5.1/sonarqube-5.1/bin/linux-x86-64
[root@localhost linux-x86-64]# ./sonar.sh console
Running SonarQube...
wrapper | --> Wrapper Started as Console
wrapper | Launching a JVM...
jvm 1 | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1 | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.
jvm 1 |
jvm 1 | 2015.05.20 02:49:20 INFO app[o.s.p.m.JavaProcessLauncher] Launch pro
java.awt.headless=true -Xmx1G -Xms256m -Xss256k -Djava.net.preferIPv4Stack=true
5 -XX:+UseCMSInitiatingOccupancyOnly -XX:+HeapDumpOnOutOfMemoryError -Djava.io.t
org.sonar.search.SearchServer /tmp/sq-process7762356701240185802properties
jvm 1 | 2015.05.20 02:49:31 INFO app[o.s.p.m.Monitor] Process[search] is up
jvm 1 | 2015.05.20 02:49:31 INFO app[o.s.p.m.JavaProcessLauncher] Launch pro
a.awt.headless=true -Dfile.encoding=UTF-8 -Djruby.management.enabled=false -Djru
utOfMemoryError -Djava.net.preferIPv4Stack=true -Djava.io.tmpdir=/tmp/sonarqube-
sonarqube-5.1/lib/jdbc/h2/h2-1.3.176.jar org.sonar.server.app.WebServer /tmp/sq-p
jvm 1 | 2015.05.20 02:50:22 INFO app[o.s.p.m.Monitor] Process[web] is up
```

- Visit `http://localhost:9000/` or `http://<IP address>:9000/`.



7. Explore **Rules** in the SonarQube dashboard.

The screenshot shows the SonarQube Rules dashboard. The left sidebar has a search bar and filters for Language (Java) and Tag (bug, CWE, convention, pitfall, cert, security, misra, brain-overload, clumsy, multi-threading). The main area displays a list of rules for Java, including:

- "equals()" should not be used to test the values of "Atomic" classes (Java, bug)
- "@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one (Java, bad-practice)
- "BigDecimal(double)" should not be used (Java, bug, cert)
- "CHECKSTYLE OFF" suppression comments should not be used (Java, bad-practice)
- "Cloneables" should implement "clone" (Java, bug)
- "compareTo" results should not be checked for specific values (Java, bug)
- "compareTo" should not return "Integer.MIN_VALUE" (Java, bug)
- "ConcurrentLinkedQueue.size()" should not be used (Java, performance)
- "deleteOnExit" should not be used (Java, performance)
- "Double.longBitsToDouble" should not be used for "int" (Java, bug)

8. Verify **Settings** in the SonarQube dashboard.

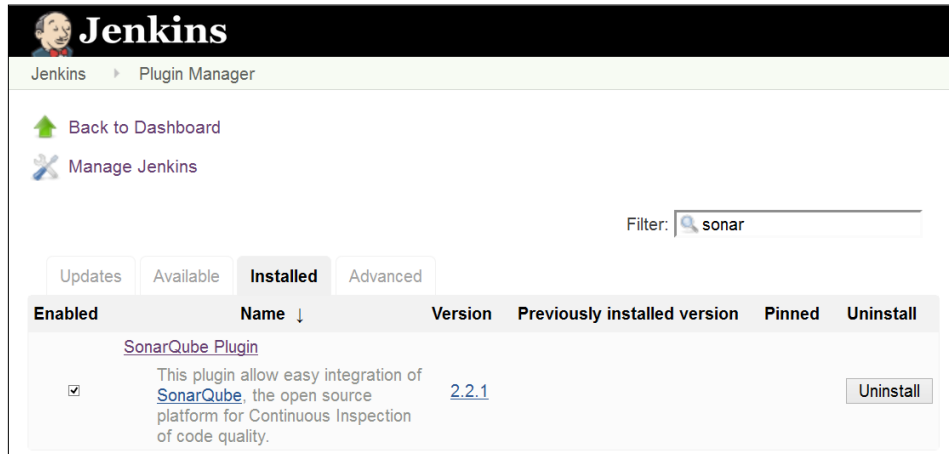
The screenshot shows the SonarQube Settings dashboard. The left sidebar has tabs for Configuration, Security, and System. The main area displays System Info, including:

- SONARQUBE
- Server ID
- Version: 5.1
- External User Authentication
 - Automatic User Creation: true
 - Allow Users to Sign Up: false
 - Force authentication: false
- Home Dir: /usr/sonarqube-5.1
- Data Dir: /usr/sonarqube-5.1/data
- Logs Dir: /usr/sonarqube-5.1/logs
- Temp Dir: /usr/sonarqube-5.1/temp

9. Create `sonar-project.properties`, and save it in a repository where the project is stored:

```
# must be unique in a given SonarQube instance
sonar.projectKey=Ant:project
# this is the name displayed in the SonarQube UI
sonar.projectName=Ant project
sonar.projectVersion=1.0
sonar.sources=src
```

10. Install the SonarQube plugin in Jenkins. To know more on this, visit <https://wiki.jenkins-ci.org/display/JENKINS/SonarQube+plugin>.



11. Click on **Manage Jenkins** and go to **Configure System**. Go to the **SonarQube** section, and configure SonarQube in Jenkins.

SonarQube installations	
Name	
Disable	☐
Server URL	http://localhost:9000/ Default is http://localhost:9000
SonarQube account login	admin SonarQube account used to perform analysis. Mandatory when anonymous access is disabled.
SonarQube account password	••••• SonarQube account used to perform analysis. Mandatory when anonymous access is disabled.
Database URL	 Do not set if default embedded database.
Database login	

12. Add Build step to **Invoke Standalone SonarQube Analysis** in a build Job.

Invoke Standalone SonarQube Analysis

Task to run

JDK

(Inherit From Job)

JDK to be used for this sonar analysis

Path to project properties

Analysis properties

JVM Options

Delete

13. Run the build job, and if you get a certificate error, execute the `svn export` command to solve the certificate issue.

```

ERROR: Error during Sonar runner execution
org.sonar.runner.impl.RunnerException: Unable to execute Sonar
    at
    org.sonar.runner.impl.BatchLauncher$1.delegateExecution(BatchLauncher.java:91)
    at org.sonar.runner.impl.BatchLauncher$1.run(BatchLauncher.java:75)
    at java.security.AccessController.doPrivileged(Native Method)
    at org.sonar.runner.impl.BatchLauncher.doExecute(BatchLauncher.java:69)
    at org.sonar.runner.impl.BatchLauncher.execute(BatchLauncher.java:50)
    at org.sonar.runner.api.EmbeddedRunner.doExecute(EmbeddedRunner.java:102)
    at org.sonar.runner.api.Runner.execute(Runner.java:100)
    at org.sonar.runner.Main.executeTask(Main.java:70)
    at org.sonar.runner.Main.execute(Main.java:59)
    at org.sonar.runner.Main.main(Main.java:53)
Caused by: java.lang.IllegalStateException: The svn blame command [svn blame --xml
--non-interactive -x -w src/com/vaannila/domain/User.java] failed: svn: OPTIONS of
'https://192.168.1.12/svn/MS/AntExample1/src/com/vaannila/domain/User.java':
authorization failed: Could not authenticate to server: rejected Basic challenge
(https://192.168.1.12)
    at org.sonar.plugins.scm.svn.SvnBlameCommand.blame(SvnBlameCommand.java:110)
    at
    org.sonar.plugins.scm.svn.SvnBlameCommand.access$000(SvnBlameCommand.java:45)
    at org.sonar.plugins.scm.svn.SvnBlameCommand$1.call(SvnBlameCommand.java:91)
    at org.sonar.plugins.scm.svn.SvnBlameCommand$1.call(SvnBlameCommand.java:88)
    at java.util.concurrent.FutureTask.run(FutureTask.java:262)
    at
    java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at
    java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:745)
ERROR:
ERROR: Re-run SonarQube Runner using the -X switch to enable full debug logging.
Build step 'Invoke Standalone SonarQube Analysis' marked build as failure
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: FAILURE

```

14. Execute the `svn export` command to solve certificate issue on a virtual machine where SonarQube and Jenkins are installed, as shown in the following screenshot:



```
root@localhost/tmp/sonarqube-5.1/sonarqube-5.1/bin/linux-x86-64
File Edit View Search Terminal Tabs Help
root@localhost/tmp root@localhost/tmp/sonar... root@localhost/tmp/sonar...
[root@localhost linux-x86-64]# svn export https://192.168.13.1/svn/MS/AntExample
1/ --username mitesh51 --password nirma51
Error validating server certificate for 'https://192.168.13.1:443':
- The certificate is not issued by a trusted authority. Use the
  fingerprint to validate the certificate manually!
- The certificate hostname does not match.
Certificate information:
- Hostname: MS
- Valid: from Thu, 14 May 2015 17:24:51 GMT until Sun, 11 May 2025 17:24:51 GMT
- Issuer: MS
- Fingerprint: ac:bb:e8:17:d1:91:06:d8:2c:e2:b4:b5:54:e3:bc:60:e5:d7:93:17
(R)ectect, accept (t)emporarily or accept (p)ermanently? p

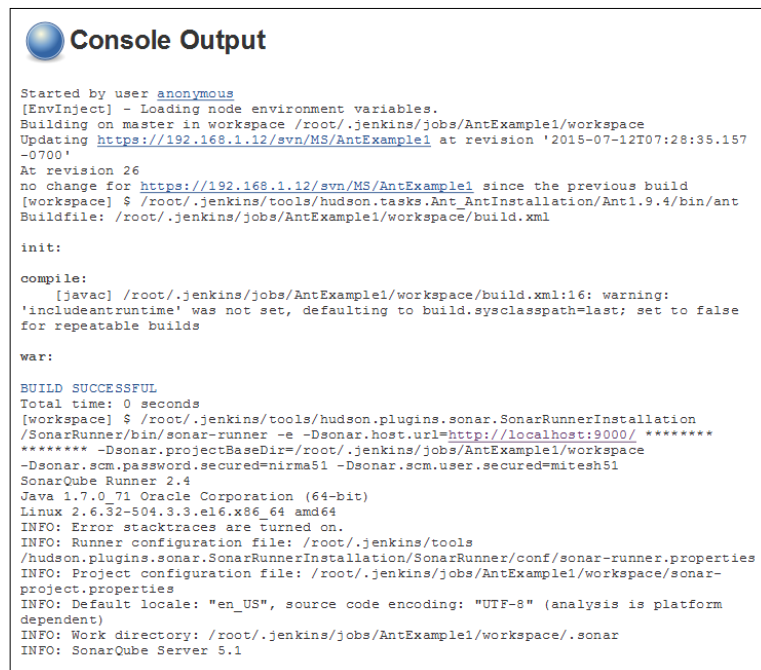
-----
ATTENTION! Your password for authentication realm:

<https://192.168.13.1:443> VisualSVN Server

can only be stored to disk unencrypted! You are advised to configure
your system so that Subversion can store passwords encrypted, if
possible. See the documentation for details.

You can avoid future appearances of this warning by setting the value
of the 'store-plaintext-passwords' option to either 'yes' or 'no' in
'/root/.subversion/servers'.
-----
Store password unencrypted (yes/no)? yes
A  AntExample1
```

15. Run the build job.



```
Console Output

Started by user anonymous
[EnvInject] - Loading node environment variables.
Building on master in workspace /root/.jenkins/jobs/AntExample1/workspace
Updating https://192.168.1.12/svn/MS/AntExample1 at revision '2015-07-12T07:28:35.157-0700'
At revision 26
no change for https://192.168.1.12/svn/MS/AntExample1 since the previous build
[workspace] $ /root/.jenkins/tools/hudson.tasks.Ant_AntInstallation/Ant1.9.4/bin/ant
Buildfile: /root/.jenkins/jobs/AntExample1/workspace/build.xml

init:

compile:
[javac] /root/.jenkins/jobs/AntExample1/workspace/build.xml:16: warning:
'inincludeantruntime' was not set, defaulting to build.sysclasspath=last; set to false
for repeatable builds

war:

BUILD SUCCESSFUL
Total time: 0 seconds
[workspace] $ /root/.jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation
/SonarRunner/bin/sonar-runner -e -Dsonar.host.url=http://localhost:9000/ *****
-Dsonar.projectBaseDir=/root/.jenkins/jobs/AntExample1/workspace
-Dsonar.scm.password.secured=nirma51 -Dsonar.scm.user.secured=mitesh51
SonarQube Runner 2.4
Java 1.7.0_71 Oracle Corporation (64-bit)
Linux 2.6.32-504.3.3.el6.x86_64 amd64
INFO: Error stacktraces are turned on.
INFO: Runner configuration file: /root/.jenkins/tools
/hudson.plugins.sonar.SonarRunnerInstallation/SonarRunner/conf/sonar-runner.properties
INFO: Project configuration file: /root/.jenkins/jobs/AntExample1/workspace/sonar-
project.properties
INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform
dependent)
INFO: Work directory: /root/.jenkins/jobs/AntExample1/workspace/.sonar
INFO: SonarQube Server 5.1
```

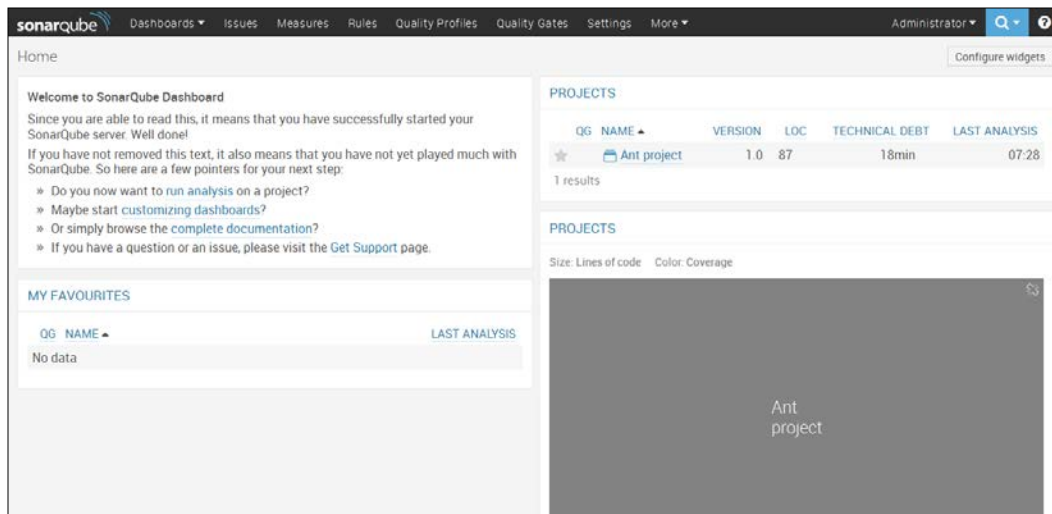
16. Verify the Sonar execution steps in the console.

```

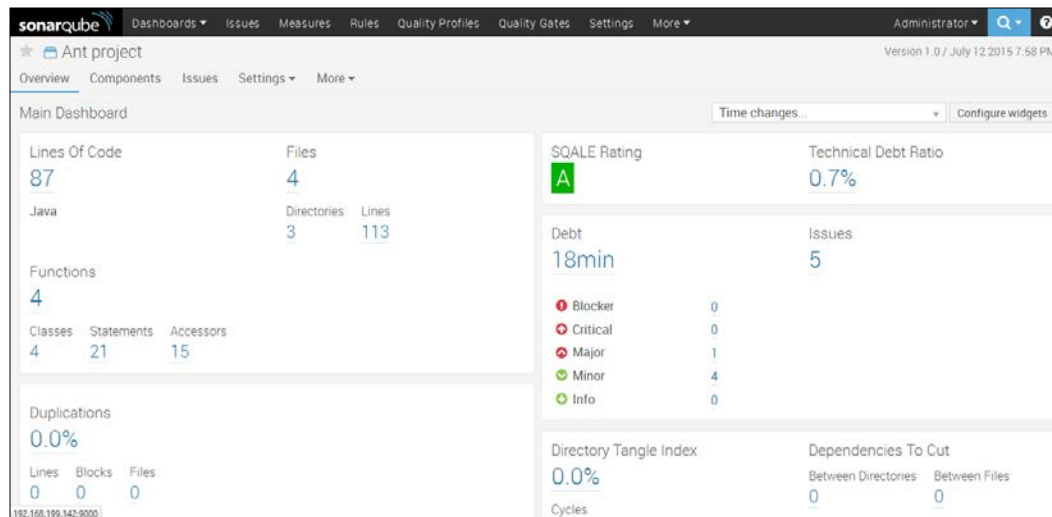
07:28:49.303 INFO - Cross-project analysis disabled
07:28:49.389 INFO - Sensor CPD Sensor (done) | time=87ms
07:28:49.390 INFO - No quality gate is configured.
07:28:49.437 INFO - Compare to previous analysis (2015-07-12)
07:28:49.444 INFO - Compare over 30 days (2015-06-12, analysis of Sun Jul 12
07:14:15 PDT 2015)
07:28:50.399 INFO - Execute decorators...
07:28:51.907 INFO - Store results in database
07:28:52.608 INFO - Analysis reports generated in 36ms, dir size=1 KB
07:28:52.622 INFO - Analysis reports compressed in 14ms, zip size=3 KB
07:28:52.716 INFO - Analysis reports sent to server in 94ms
07:28:52.716 INFO - ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard/index/Ant:project
07:28:52.716 INFO - Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report.
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
Total time: 15.545s
Final Memory: 13M/124M
INFO: -----
Deploying /root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war to container Tomcat 7.x Remote
  Redeploying [/root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war]
  Undeploying [/root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war]
  Deploying [/root/.jenkins/jobs/AntExample1/workspace/dist/AntExample.war]
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: SUCCESS

```

17. Refresh the dashboard of SonarQube, and we will be able to see details on the recently executed build in SonarQube, as shown in the following screenshot:



18. To get more details on code verification, click on the project, and we will be able to get details on **Lines of Code**, **Duplications**, **Complexity**, and so on.



Explore more things on SonarQube and Jenkins integration, as in the following steps.

Exploring Static Code Analysis Plugins

Static Code Analysis Plugins provide utilities for the static code analysis plugins. Jenkins interprets the result files of several static code analysis tools with the use of different plugins for configuration and parsing. We can have more flexibility with these plugins to build exactly what you want.

To install any of these plugins, go to the Jenkins dashboard, click on **Manage Jenkins**, and select the **Manage Plugins** link. Go to the **Available** tab, find the respective plugin, and select it. Click on **Download now**, and install after restart.

All these results are visualized by the same backend. The following plugins use the same visualization:

Checkstyle Plugin

The Checkstyle plugin generates the report for an open source static code analysis program, Checkstyle.

To know more about the Checkstyle plugin, visit <https://wiki.jenkins-ci.org/display/JENKINS/Checkstyle+Plugin>.

FindBugs Plugin

The FindBugs plugin is supported by the Static Analysis Collector plugin that shows the results in aggregated trend graphs, health reporting, and builds stability.

To learn more about this, visit <https://wiki.jenkins-ci.org/display/JENKINS/FindBugs+Plugin>.

Compiler Warnings Plugin

The Compiler Warnings plugin generates the trend report for compiler warnings in the console log, or in log files.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin>.

To publish the combined results of Checkstyle, FindBugs, and compiler warnings plugins, go to the **Build** section of any job, and click on **Add post-build action** and select **Publish combined analysis results**.

Publish combined analysis results

Checkstyle warnings ☒

FindBugs warnings ☒

Compiler warnings ☒

Run always ☐

By default, this plug-in runs only for stable or unstable builds, but not for failed builds. If this plug-in should run even for failed builds then activate this check box.

Health thresholds 100% 0%

Configure the thresholds for the build health. If left empty then no health report is created. If the actual number of warnings is between the provided thresholds then the build health is interpolated.

Health priorities ☐ Only priority high ☐ Priorities high and normal ☒ All priorities

Determines which warning priorities should be considered when evaluating the build health.

Status thresholds (Totals)	All priorities	Priority high	Priority normal	Priority low

If the number of total warnings is greater than one of these thresholds then a build is considered as unstable or failed, respectively. I.e., a value of 0 means that the build status is changed if there is at least one warning found. Leave this field empty if the state of the build should not depend on the number of warnings.

We can also see these results with the use of the Dashboard View plugin.

In the configuration of a Dashboard view, click on **Edit View** and select checkboxes in the **Number of warnings** section. Add **Dashboard Portlets** in different sections for Checkstyle, Compiler, and Findbug.

Portlets at the top of the page

Checkstyle warnings per project

Name

Hide zero warnings projects ☐

Delete

Compiler warnings per project

Name

Hide zero warnings projects ☐

Parser

Select the parser whose warnings should be shown.

Delete

Add Dashboard Portlet to the top of the view

Verify the view after all the changes and running build jobs.

Jenkins

search

ENABLE AUTO REFRESH

Jenkins > MyView >

add description

New Item

People

Build History

Edit View

Delete View

Project Relationship

Check File Fingerprint

Manage Jenkins

Credentials

Disk usage

Jenkins 100K

Build Queue

No builds in the queue.

Build Executor Status

192.168.199.142:8080

AllMy MonitoringMyViewPetClinicPublicCloudDeploymentTest-Pipelineset-pipeline+

S	W	Name ↓	Last Success	Last Duration	# Warnings
		AntExample1	13 min - #12	21 sec	0
		CounterApp	1 mo 9 days - #23	57 sec	3
		PetClinic-Test	17 days - #6	18 min	0

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Checkstyle warnings per project

Compiler warnings per project

Job ↓	Total	High	Normal	Low
AntExample1	-	-	-	-
CounterApp	3	0	3	0
PetClinic-Test	-	-	-	-
Total	3	0	3	0

The following plugins are also useful.

DRY Plugin

The DRY plugin shows the duplicate code blocks in your project. It only shows the results of duplicate code checker tools.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/DRY+Plugin>.

PMD Plugin

The PMD plugin scans the `pmd.xml` files in the build workspace, and reports warnings.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/PMD+Plugin>.

Task Scanner Plugin

The Task Scanner plugin scans the workspace files for open tasks and provides a trend report.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/Task+Scanner+Plugin>.

CCM Plugin

The CCM plugin provides details on cyclomatic complexity for .NET code.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/CCM+Plugin>.

Android Lint Plugin

The Android Lint plugin parses the output from the Android lint tool.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/Android+Lint+Plugin>.

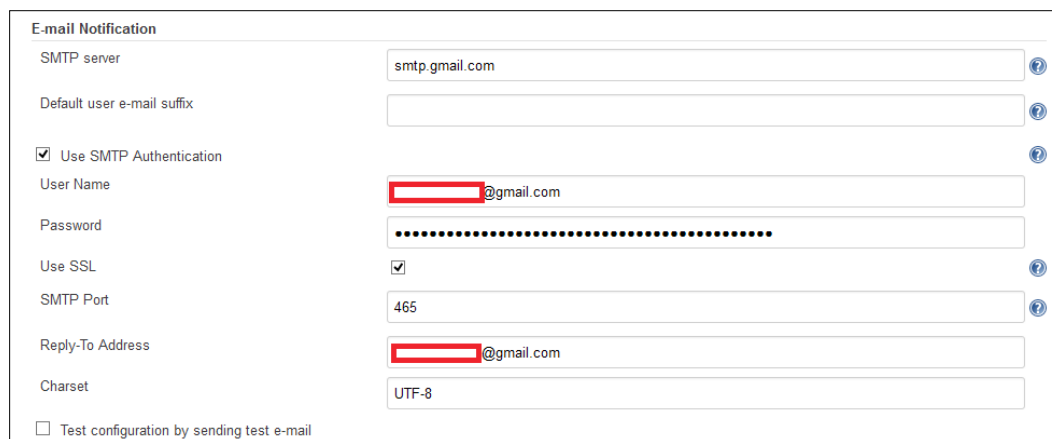
OWASP Dependency-Check Plugin

The Dependency-Check Jenkins Plugin features the ability to perform a dependency analysis build.

To know more, visit <https://wiki.jenkins-ci.org/display/JENKINS/OWASP+Dependency-Check+Plugin>.

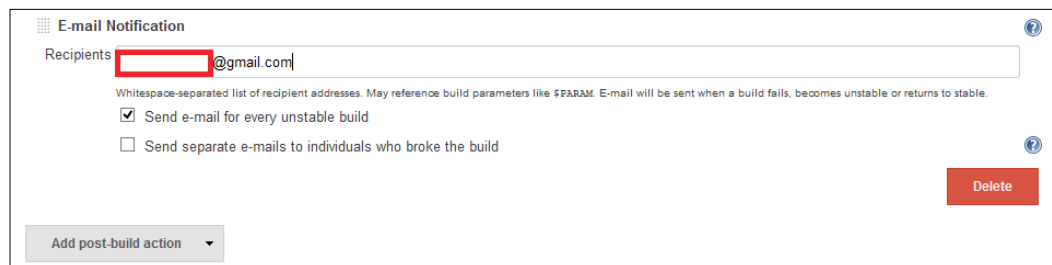
E-mail notifications on build status

To send an e-mail notification based on build status, we need to configure SMTP details. Click on **Manage Jenkins**, and go to **Configure System**. Go to the **E-mail Notification** section.



The screenshot shows the 'E-mail Notification' configuration page in Jenkins. It includes fields for SMTP server (smtp.gmail.com), Default user e-mail suffix, User Name (redacted@gmail.com), Password (masked with dots), Use SSL (checked), SMTP Port (465), Reply-To Address (redacted@gmail.com), and Charset (UTF-8). There is a checkbox for 'Test configuration by sending test e-mail' at the bottom.

Go to build Job configuration, and click on **Add post-build action**. Select **E-mail Notification**. Provide the recipients list and save.



The screenshot shows the 'E-mail Notification' configuration page within a job configuration. It includes a 'Recipients' field (redacted@gmail.com), a checkbox for 'Send e-mail for every unstable build' (checked), and a checkbox for 'Send separate e-mails to individuals who broke the build' (unchecked). There is a 'Delete' button and an 'Add post-build action' dropdown at the bottom.

Run the build job, and a broken build will result in an e-mail notification in the mailbox.

Self-test questions

Q1. Which languages are supported by SonarQube?

1. Java
2. C#
3. PHP
4. Python
5. C/C++
6. JavaScript
7. All of the above

Q2. Which among these is not a Static Code Analysis plugin?

1. DRY Plugin
2. PMD Plugin
3. Task Scanner Plugin
4. FindBugs Plugin
5. None of the above

Summary

Here again, we are at the end of another chapter. We need to remember that every new beginning comes from some other beginning's end. To summarize, we learned how to manage code quality of applications configured, and how to use notification features to send information to developers based on the failed build. We also covered some static code analysis plugins in short, to get some idea about it. In the next chapter, we will learn how to manage and monitor Jenkins.

7

Managing and Monitoring Jenkins

"Fall in the beginning + Fall often + Learn to recover quickly = Faster time to market"

– Anonymous

We learned Sonar integration with Jenkins, an overview of static code analysis plugins, and notification of build status in the last chapter. Now, it's time to focus on management and monitoring of Jenkins.

This chapter gives insight into management of Jenkins nodes and monitoring of them with Java Melody to provide details on utilization of resources. It also covers how to manage and monitor build jobs. This chapter describes basic security configuration in detail that is available in Jenkins for a better access control and authorization. The following is the list of topics that we will cover in this chapter:

- Managing Jenkins master and slave nodes
- Jenkins monitoring with JavaMelody
- Managing disk usage
- Build job-specific monitoring with the Build Monitor plugin
- Managing access control and authorization
- Maintaining role and project-based security
- Managing an admin account
- Audit Trail Plugin – an overview and usage

Managing Jenkins master and slave nodes

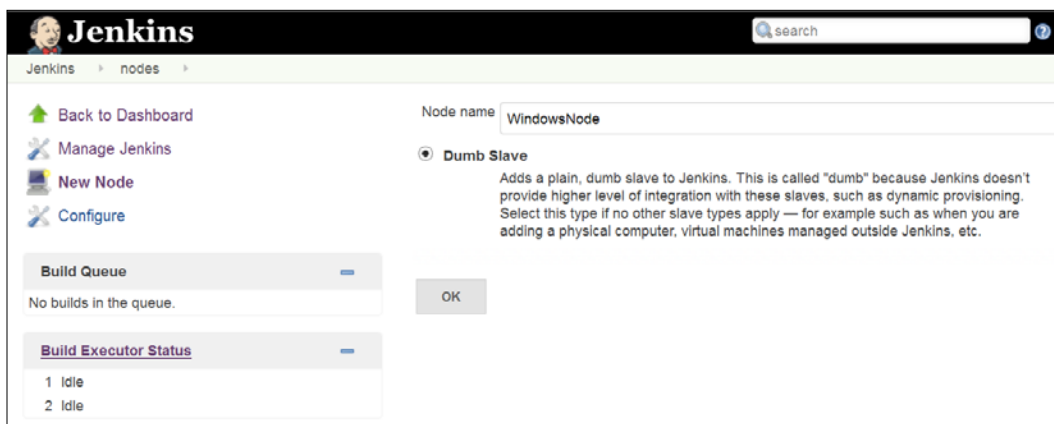
A master represents basic installation of Jenkins and handles all tasks for the build system. It can satisfy all user requests and has the capacity to build projects on its own. A slave is a system that is set up to reduce the burden of build projects from the master but delegation behavior depends on the configuration of each project. Delegation can be configured specifically to build job.

1. On the Jenkins dashboard, go to **Manage Jenkins**. Click on **Manage Nodes** link. It will provide information on all nodes, as shown in the following screenshot:

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Linux (amd64)	In sync	5.86 GB	1.94 GB	5.86 GB	0ms 
	Data obtained	48 sec	48 sec	48 sec	48 sec	48 sec	48 sec

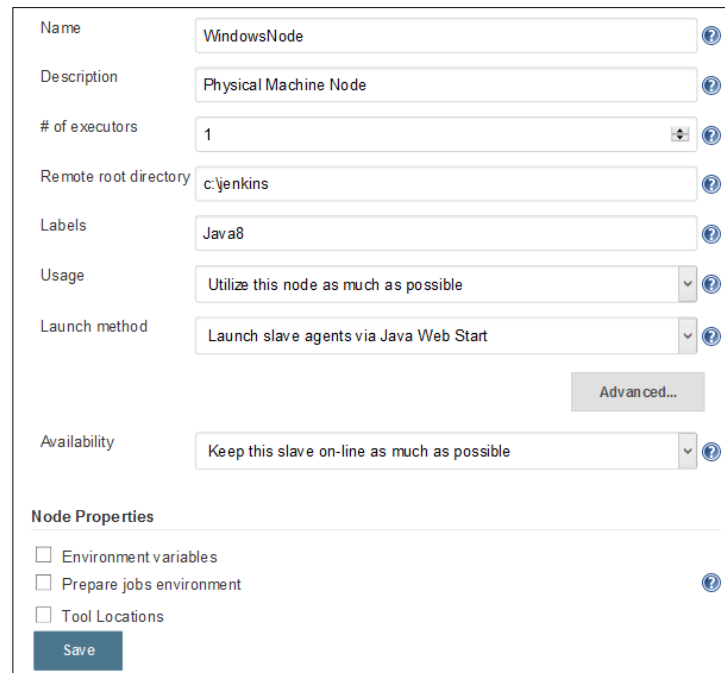
Refresh status

2. To create a slave node, click on **New Node**.



The screenshot shows the Jenkins 'New Node' configuration page. On the left, there is a sidebar with links: 'Back to Dashboard', 'Manage Jenkins', 'New Node' (highlighted), and 'Configure'. Below these links are two sections: 'Build Queue' showing 'No builds in the queue.' and 'Build Executor Status' showing two 'Idle' executors. The main area is for configuring a new node. It has a 'Node name' field with 'WindowsNode' entered. Below this is a radio button selection for 'Dumb Slave', which is selected. A description for 'Dumb Slave' explains it's a plain slave with no dynamic provisioning. An 'OK' button is at the bottom right.

3. Provide **Name**, **Description**, **Labels** and so on. Select **Launch slave agents via Java Web Start** as **Launch method**. Provide **Labels**; in our case, it is java8:



Name: WindowsNode

Description: Physical Machine Node

of executors: 1

Remote root directory: c:\jenkins

Labels: Java8

Usage: Utilize this node as much as possible

Launch method: Launch slave agents via Java Web Start

Advanced...

Availability: Keep this slave on-line as much as possible

Node Properties

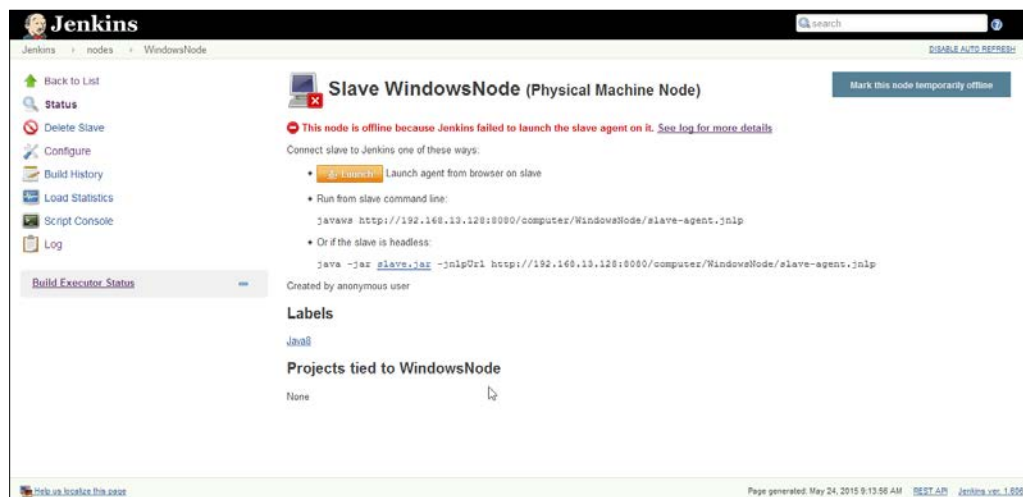
☐ Environment variables

☐ Prepare jobs environment

☐ Tool Locations

Save

4. Click on **Save**. It will open a page that gives details on how to launch the slave node.



Jenkins

nodes > WindowsNode

Slave WindowsNode (Physical Machine Node)

Mark this node temporarily offline

This node is offline because Jenkins failed to launch the slave agent on it. [See log for more details](#)

Connect slave to Jenkins one of these ways:

- Launch agent from browser on slave
- Run from slave command line:


```
java http://192.168.13.128:8080/computer/WindowsNode/slave-agent.jnlp
```
- Or if the slave is headless:


```
java -jar slave.jar -jnlpUrl http://192.168.13.128:8080/computer/WindowsNode/slave-agent.jnlp
```

Created by anonymous user

Labels

Java8

Projects tied to WindowsNode

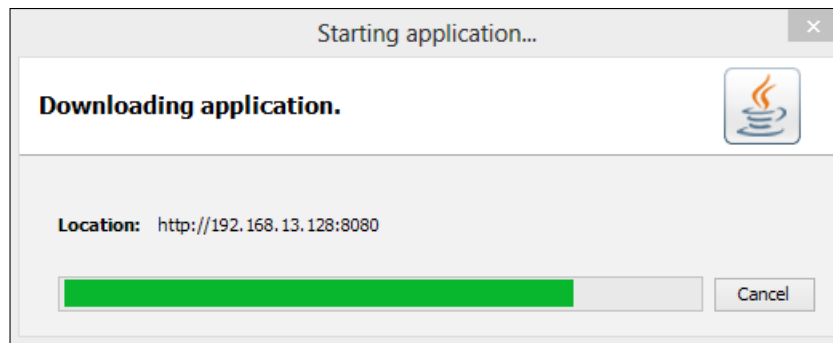
None

Page generated: May 24, 2015 9:13:56 AM

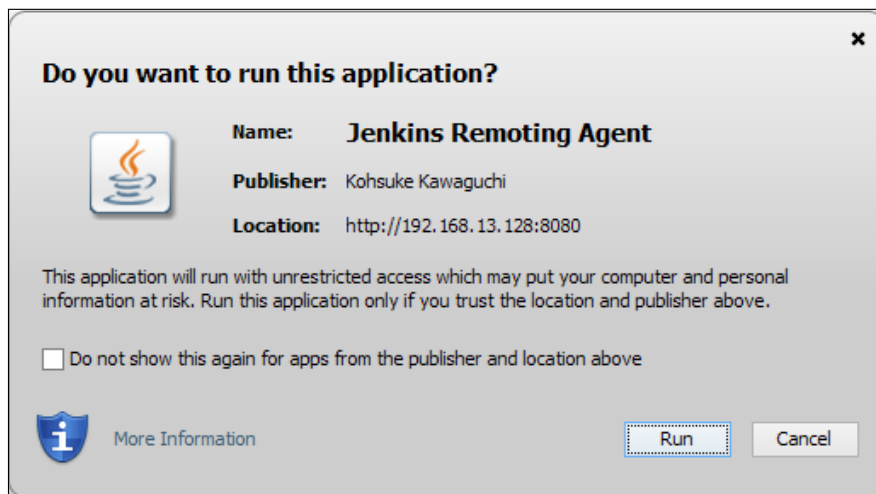
5. Open terminal on the Windows machine and run `javaws http://192.168.13.128:8080/computer/WindowsNode/slave-agent.jnlp`.



It will open a dialogue box for downloading the application.



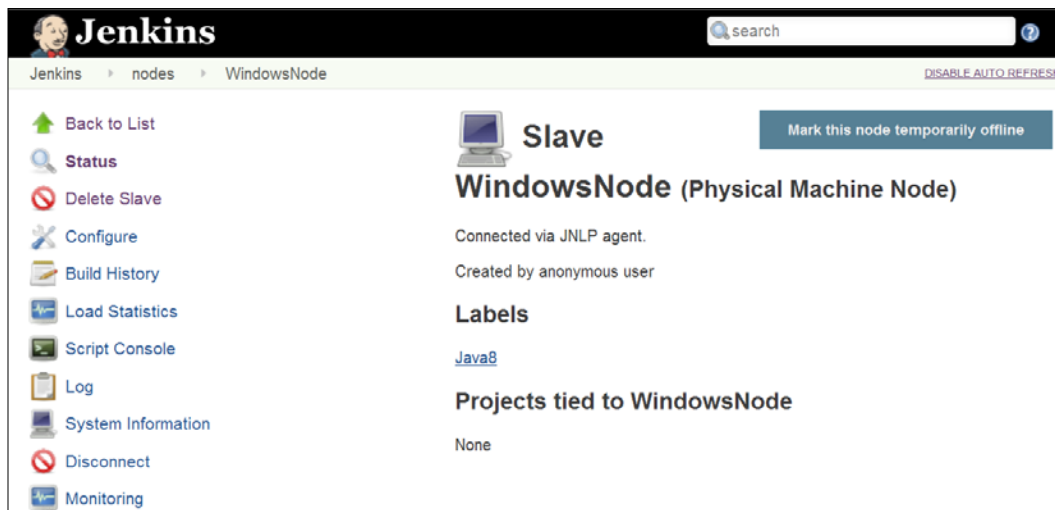
6. Run Jenkins Remoting Agent.



A small window for the Jenkins slave agent will open.



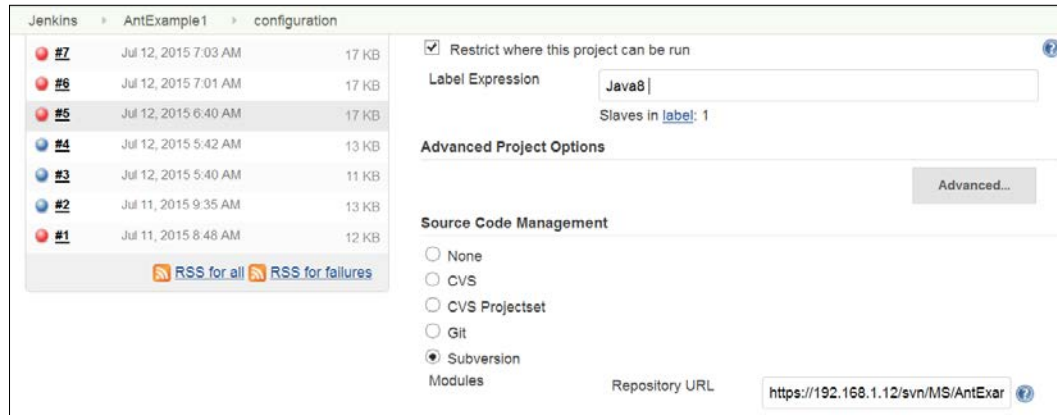
The **slave WindowsNode** will be connected via the JNLP agent.



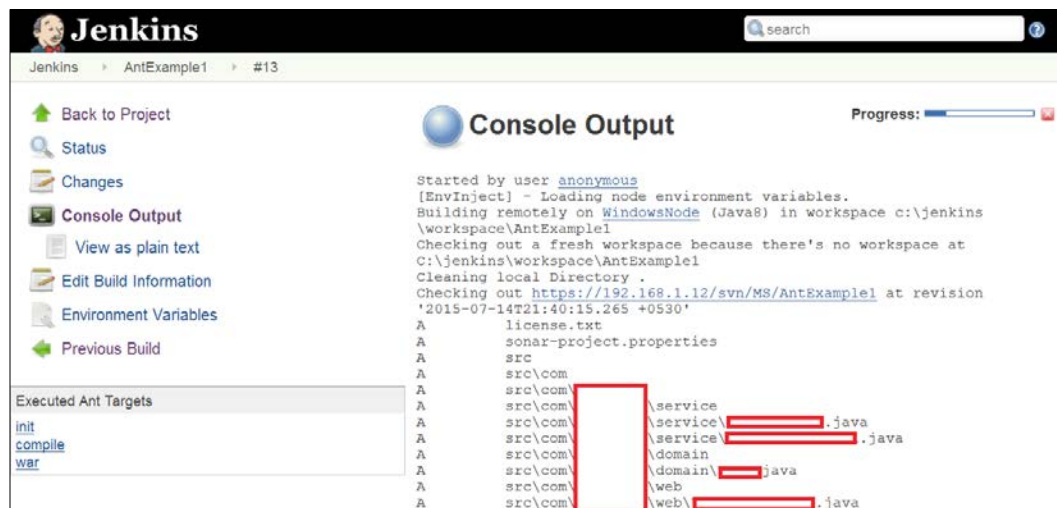
- On the Jenkins dashboard, go to **Manage Jenkins**. Click on the **Manage Nodes** link. It will provide information on all nodes, as shown in the following screenshot. Verify both the nodes in the **Build Executor Status** section of the leftmost sidebar.

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	master	Linux (amd64)	In sync	5.86 GB	1.94 GB	5.86 GB	0ms	
	WindowsNode	Windows 8 (amd64)	In sync	215.13 GB	4.27 GB	215.13 GB	3340ms	
	Data obtained	42 sec	42 sec	42 sec	42 sec	42 sec	42 sec	
Refresh status								

8. If we want to run a selective build job on to a specific node, then we can configure it build job-wise, as shown in the following screenshot. Check **Restrict where this project can be run** and provide **Label Expression** given to the specific node on the job configuration page.



9. Click on **Build Now** to execute build. Verify the console and find building remotely on WindowsNode we configured in the preceding section. It will check out the code on slave and perform operations on the specific node only.

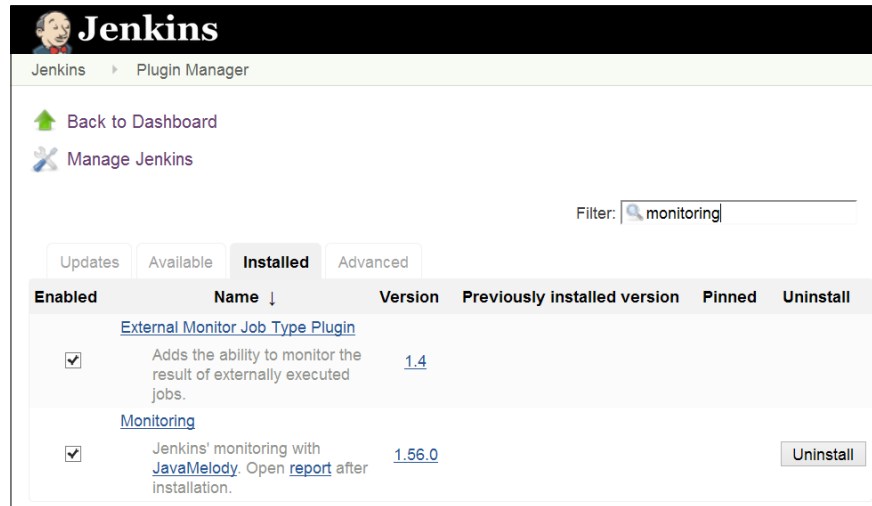


Such configuration is useful where we want to run build job in a specific set of runtime environment, which is available on the specific node.

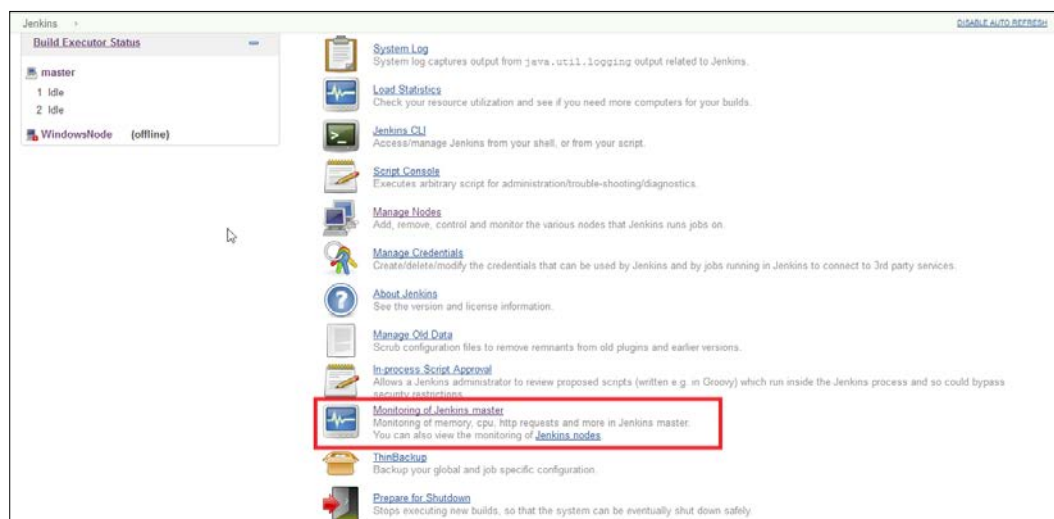
Jenkins monitoring with JavaMelody

The Monitoring plugin provides monitoring of Jenkins with JavaMelody.

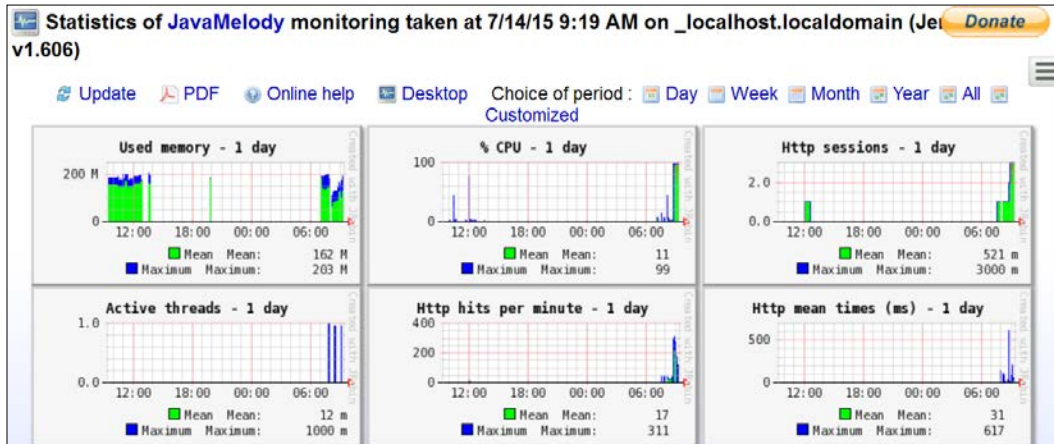
It provides charts of a CPU, memory, system load average, HTTP response time, and so on. It also provides details of HTTP sessions, errors and logs, actions for GC, heap dump, invalidate session(s), and so on. Install the Monitoring plugin from the Jenkins Dashboard.



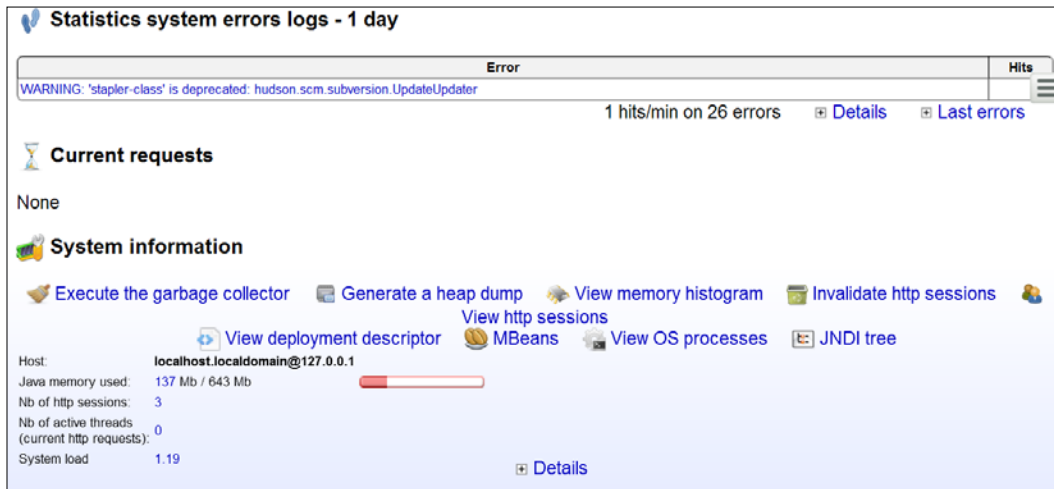
1. On the Jenkins dashboard, click on **Manage Jenkins**. Click on **Monitoring of Jenkins master**, as shown in the following screenshot:



- It will open the statistics of JavaMelody monitoring, as shown in the following screenshot. Observe all statistics:



- Scroll down the page and we will find **Statistics system errors logs**.



- To get more information, click on the **Details** link of any section. Statistics of HTTP are as shown in the following figure:

 **Statistics http - 1 day**

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error	Mean size (Kb)
http global	100	5,774	24	21,126	415	100	7	0.03	1
http warning	10	24	640	3,100	736	22	410	0.00	25
http severe	57	14	5,840	21,126	6,132	46	1,433	14.29	196

44 hits/min on 290 requests [Details](#)

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of cumulative cpu time	Mean cpu time (ms)	% of system error	Mean size (Kb)
/descriptorByName/com.cloudbees.jenkins.GitHubPushTrigger/checkHookUrl ajax GET	14	2	10,583	21,126	14,909	0	17	0.00	3
/ ajax GET	13	3	6,321	6,321	6,787	6	882	66.67	18
/configure GET	13	2	9,369	10,498	1,595	18	4,078	0.00	296
/job/AntExample1/configure GET	10	3	5,183	9,984	4,170	13	1,986	0.00	655
/ GET	3	9	562	3,100	954	8	421	0.00	41
/manage GET	3	7	693	2,409	894	6	386	0.00	20
/computer/ GET	3	60	75	1,284	166	5	39	0.00	12
/pluginManager/installed GET	2	2	1,797	2,379	823	2	610	0.00	62
/configSubmit POST	2	4	826	923	174	4	457	0.00	0

- Explore more at <https://wiki.jenkins-ci.org/display/JENKINS/Monitoring> to get more details on the Monitoring plugin.

Managing disk usage

- Disk Usage Plugin records disk usage. Install **Disk Usage Plugin** from the Jenkins dashboard.

 **Jenkins**

Jenkins ▸ Plugin Manager

 [Back to Dashboard](#)

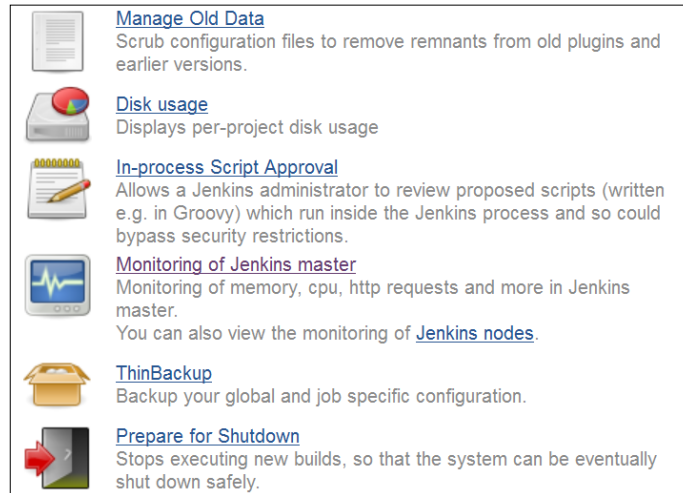
 [Manage Jenkins](#)

Filter:

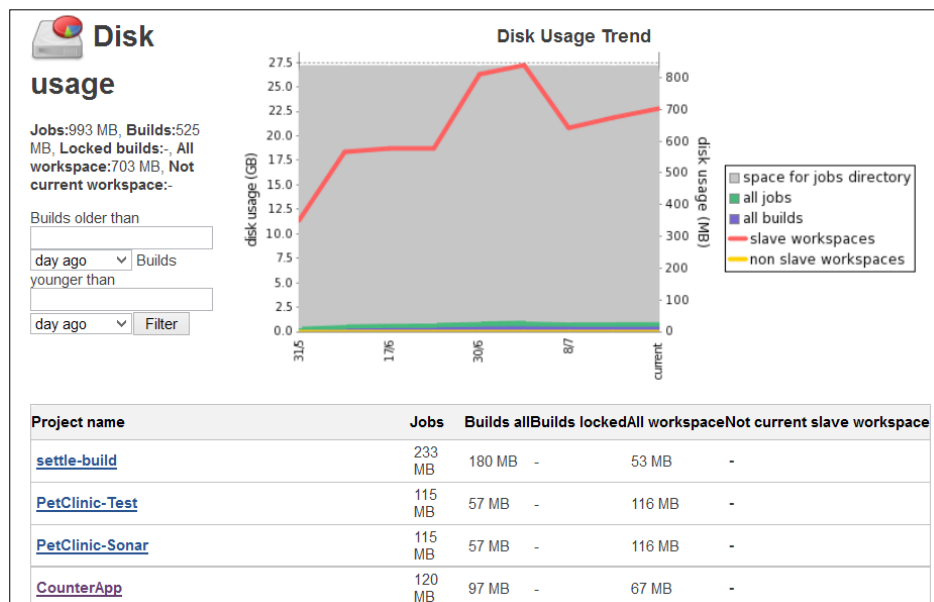
Updates Available **Installed** Advanced

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
☒	disk-usage plugin	0.25			Uninstall
	This plugin counts disk usage.				

2. Once the plugin is successfully installed, we will get the **Disk usage** link on the Manage Jenkins page, as shown in the following screenshot:



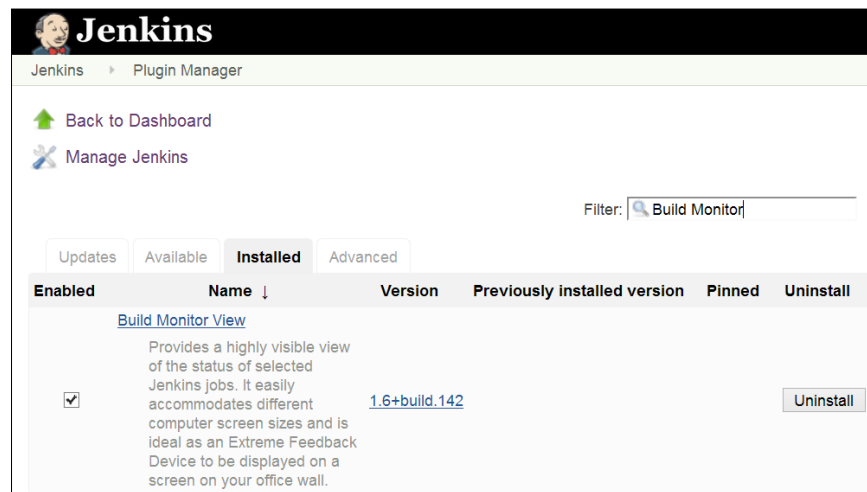
3. The Disk Usage plugin will show project-wise details for all jobs and all workspace. It will also display **Disk Usage Trend**.



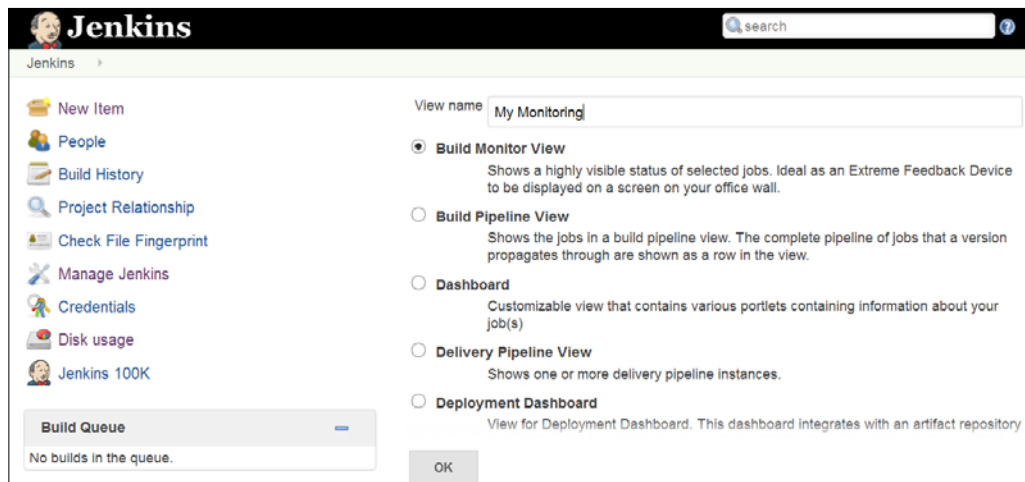
To get more details on Disk usage plugin, visit <https://wiki.jenkins-ci.org/display/JENKINS/Disk+Usage+Plugin>.

Build monitoring with Build Monitor Plugin

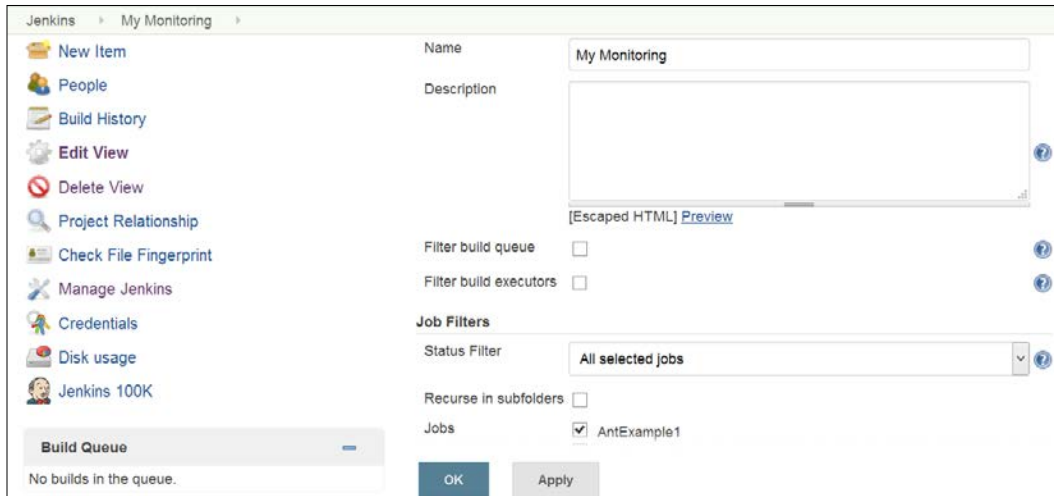
Build Monitor Plugin provides a detailed view of the status of selected Jenkins jobs. It provides the status and progress of selected jobs and names of people who might be responsible for "breaking the build". This plugin supports the Claim plugin, View Job Filters, Build Failure Analyzer, and CloudBees Folders plugin.



1. The Dashboard View plugin will be used for creating a view that provides details on build job-specific monitoring. Create a new view and select **Build Monitor View**.

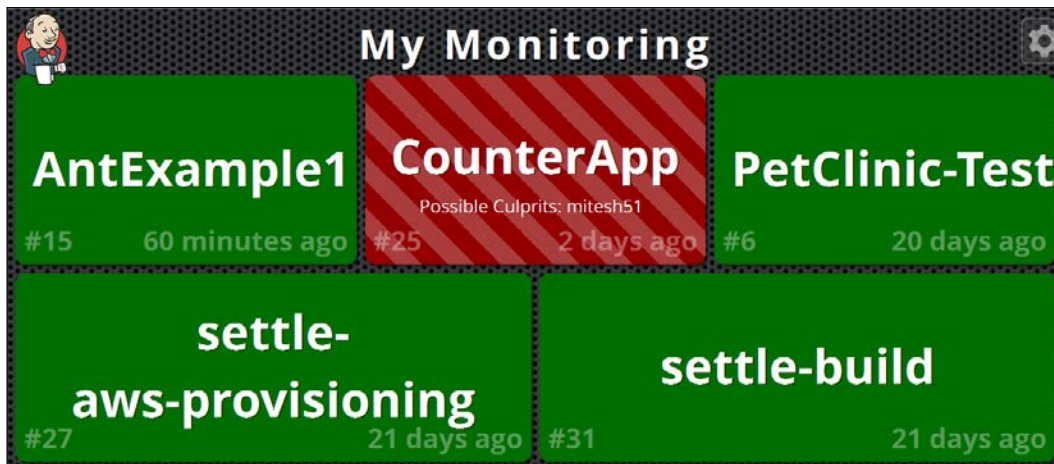


2. Select **Jobs** and save the details.



The screenshot shows the Jenkins 'My Monitoring' configuration page. On the left is a sidebar with navigation links: New Item, People, Build History, Edit View, Delete View, Project Relationship, Check File Fingerprint, Manage Jenkins, Credentials, Disk usage, and Jenkins 100K. Below these is a 'Build Queue' section showing 'No builds in the queue.' The main area contains configuration fields: 'Name' (My Monitoring), 'Description' (a text area), and 'Filter build queue' (checkbox). Below these are 'Job Filters' including 'Status Filter' (All selected jobs), 'Recurse in subfolders' (checkbox), and 'Jobs' (a list with 'AntExample1' selected). 'OK' and 'Apply' buttons are at the bottom right.

3. Click on the newly created view, and we will get a similar type of screen as given in the following screenshot:

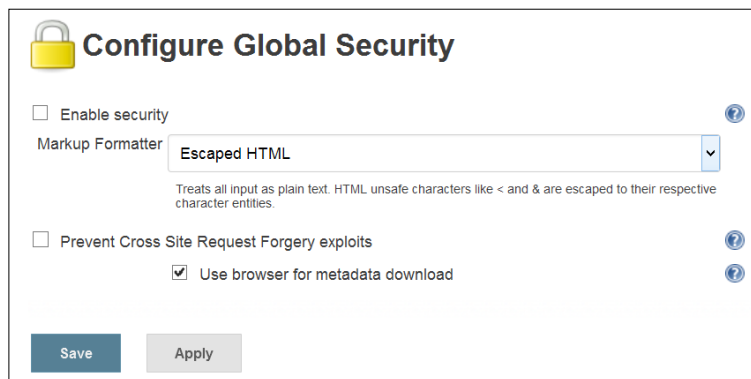


To get more details on plugin, visit <https://wiki.jenkins-ci.org/display/JENKINS/Build+Monitor+Plugin>.

Managing access control and authorization

Jenkins supports several security models, and can integrate with different user repositories.

1. Go to the Jenkins dashboard, click on **Manage Jenkins**, and click on **Configure Global Security**.
2. Click on **Enable security**.



Configure Global Security

☐ Enable security

Markup Formatter: **Escaped HTML**

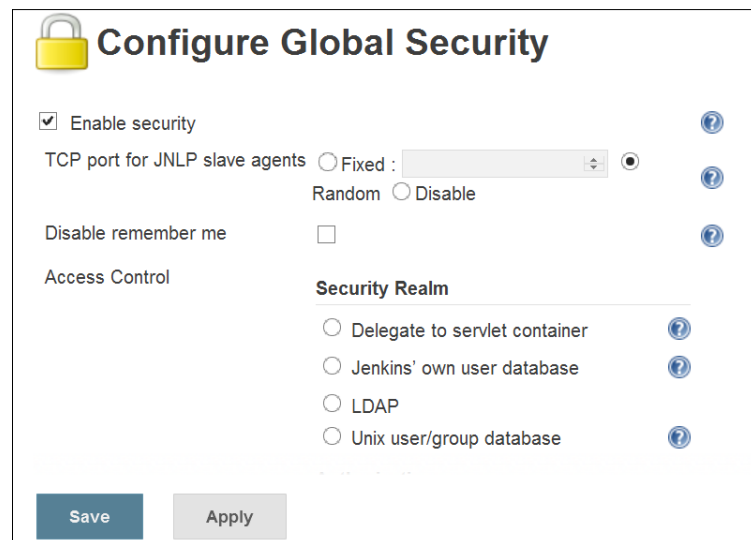
Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

☐ Prevent Cross Site Request Forgery exploits

☒ Use browser for metadata download

Save **Apply**

All options will be visible once we enable security, as shown in the following screenshot:



Configure Global Security

☒ Enable security

TCP port for JNLP slave agents: ☐ Fixed : ☒ Random ☐ Disable

Disable remember me: ☐

Access Control: **Security Realm**

☐ Delegate to servlet container

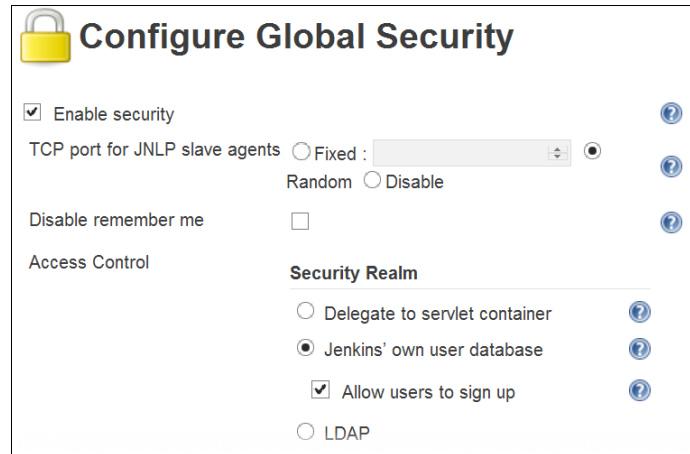
☐ Jenkins' own user database

☐ LDAP

☐ Unix user/group database

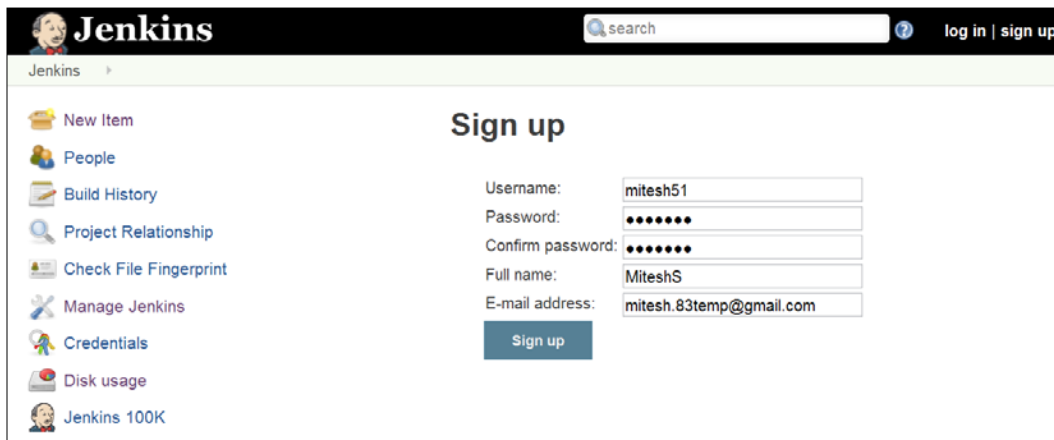
Save **Apply**

3. Click on **Jenkins' own user database**. Click on **Save**.



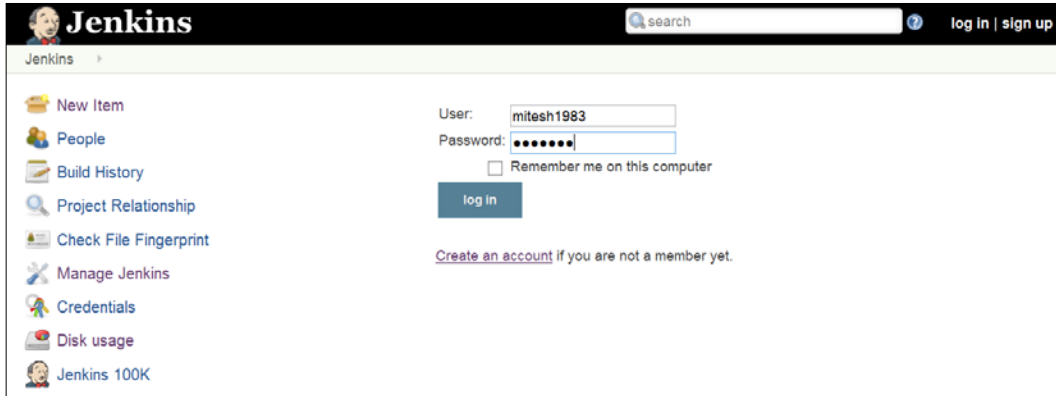
The image shows the 'Configure Global Security' dialog box in Jenkins. It has a yellow padlock icon at the top left. The title is 'Configure Global Security'. There are several settings: 'Enable security' is checked; 'TCP port for JNLP slave agents' has 'Fixed' selected with a value of 8080, and 'Random' and 'Disable' are unselected; 'Disable remember me' is unchecked; 'Access Control' is set to 'Security Realm'. Under 'Security Realm', 'Delegate to servlet container' is unselected, 'Jenkins' own user database' is selected, 'Allow users to sign up' is checked, and 'LDAP' is unselected. There are help icons (question marks) next to several settings.

4. Now, click on the **sign up** link on the top-right corner. Provide **Username**, **Password**, **Full name**, and **E-mail address**.



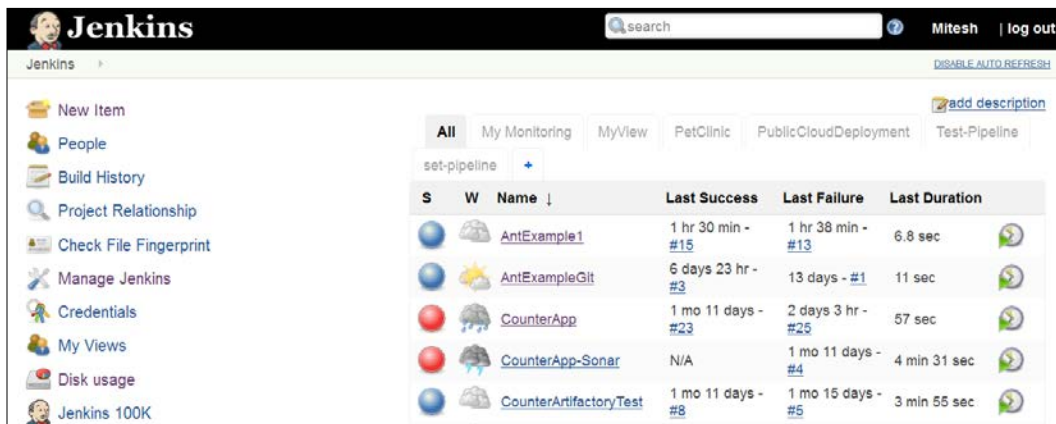
The image shows the Jenkins 'Sign up' page. The top header has the Jenkins logo, a search bar, and 'log in | sign up' links. The left sidebar contains links: 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'Credentials', 'Disk usage', and 'Jenkins 100K'. The main content area is titled 'Sign up' and contains a form with the following fields: 'Username' (filled with 'mitesh51'), 'Password' (filled with dots), 'Confirm password' (filled with dots), 'Full name' (filled with 'MiteshS'), and 'E-mail address' (filled with 'mitesh.83temp@gmail.com'). A 'Sign up' button is at the bottom of the form.

- Click on the **log in** link on the dashboard.



The image shows the Jenkins login page. At the top, there is a search bar and links for 'log in' and 'sign up'. On the left, there is a sidebar with various links: 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'Credentials', 'Disk usage', and 'Jenkins 100K'. In the center, there is a login form with fields for 'User:' (containing 'mitesh1983') and 'Password:' (masked with dots). Below the password field is a checkbox for 'Remember me on this computer' and a 'log in' button. At the bottom, there is a link that says 'Create an account if you are not a member yet.'

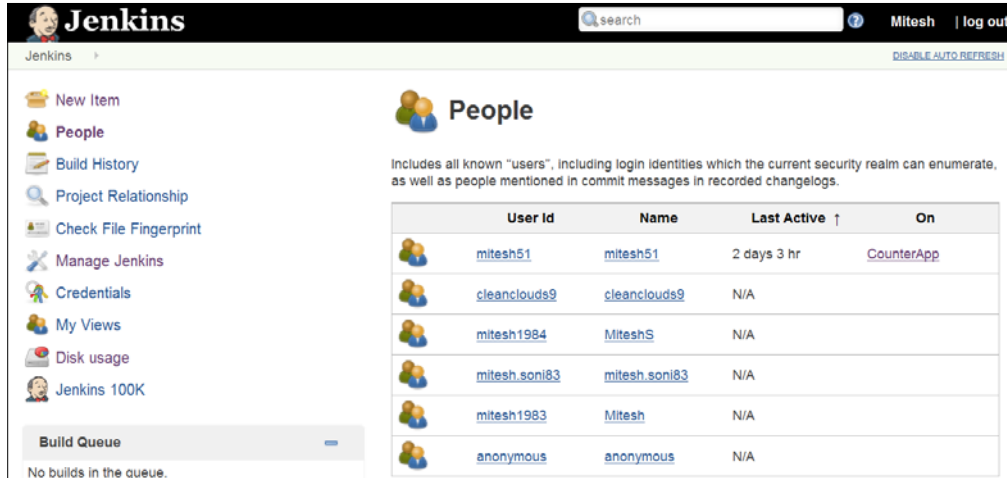
We will get the Jenkins dashboard with the username in the top-right corner.



The image shows the Jenkins dashboard after logging in. The top right corner now displays the username 'Mitesh' and a 'log out' link. Below the search bar, there is a 'DISABLE AUTO REFRESH' link and an 'add description' button. The main content area shows a list of jobs. Above the list, there are tabs for 'All', 'My Monitoring', 'MyView', 'PetClinic', 'PublicCloudDeployment', and 'Test-Pipeline'. Below these tabs, there is a 'set-pipeline' button with a plus icon. The table below lists the jobs with their status, name, last success, last failure, and last duration.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		AntExample1	1 hr 30 min - #15	1 hr 38 min - #13	6.8 sec
		AntExampleGit	6 days 23 hr - #3	13 days - #1	11 sec
		CounterApp	1 mo 11 days - #23	2 days 3 hr - #25	57 sec
		CounterApp-Sonar	N/A	1 mo 11 days - #4	4 min 31 sec
		CounterArtifactoryTest	1 mo 11 days - #8	1 mo 15 days - #5	3 min 55 sec

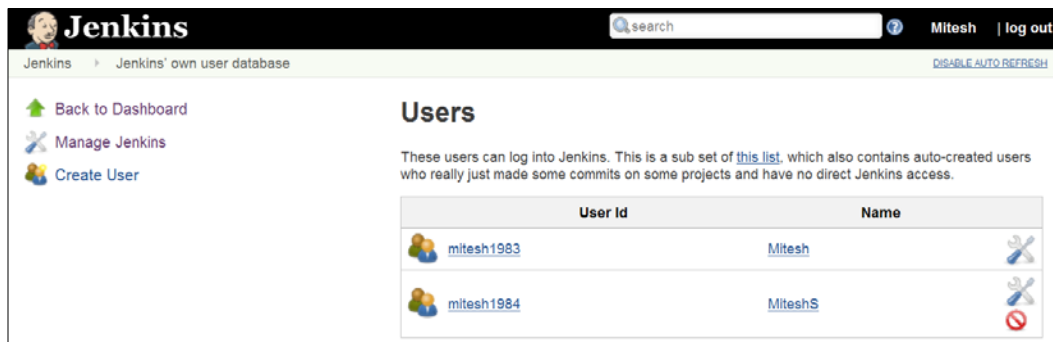
- Click on **People** to verify all users.



The screenshot shows the Jenkins 'People' page. The left sidebar contains links for New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, Credentials, My Views, Disk usage, and Jenkins 100K. The main content area is titled 'People' and includes a description: 'Includes all known "users", including login identities which the current security realm can enumerate, as well as people mentioned in commit messages in recorded changelogs.' Below this is a table with the following data:

User Id	Name	Last Active ↑	On
mitesh51	mitesh51	2 days 3 hr	CounterApp
cleanclouds9	cleanclouds9	N/A	
mitesh1984	MiteshS	N/A	
mitesh_soni83	mitesh_soni83	N/A	
mitesh1983	Mitesh	N/A	
anonymous	anonymous	N/A	

- On the Jenkins dashboard, click on **Manage Jenkins**. Click on **Manage Users**.



The screenshot shows the Jenkins 'Users' page. The left sidebar contains links for Back to Dashboard, Manage Jenkins, and Create User. The main content area is titled 'Users' and includes a description: 'These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.' Below this is a table with the following data:

User Id	Name
mitesh1983	Mitesh
mitesh1984	MiteshS

We can edit user details on the same page. This is a subset of users, which also contains auto-created users.

Maintaining roles and project-based security

For authorization, we can define **Matrix-based security** on the **Configure Global Security** page.

1. Add group or user and configure security based on different sections such as **Credentials**, **Slave**, **Job**, and so on.
2. Click on **Save**.

Authorization

☐ Anyone can do anything

☐ Legacy mode

☐ Logged-in users can do anything

☒ Matrix-based security

User/group	Overall						Credentials				Slave										
	Administer	Configure	Update	Center	Read	Run	Scripts	Upload	Plugins	Create	Delete	Manage	Domains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

User/group to add:

☐ Project-based Matrix Authorization Strategy

Escaped HTML

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

We can use multiple users for matrix-based security, as shown in the following screenshot:

Authorization

☐ Anyone can do anything

☐ Legacy mode

☐ Logged-in users can do anything

☒ Matrix-based security

User/group	Overall						Credentials				Slave										
	Administer	Configure	Update	Center	Read	Run	Scripts	Upload	Plugins	Create	Delete	Manage	Domains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
 mitesh1983	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒

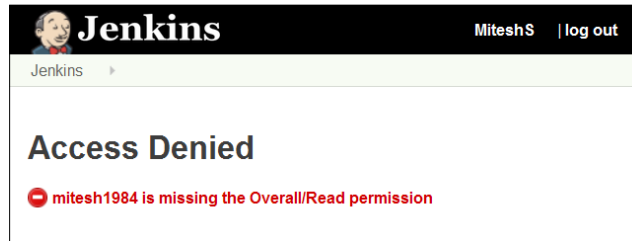
User/group to add:

☐ Project-based Matrix Authorization Strategy

Escaped HTML

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

- Try to access the Jenkins dashboard with a newly added user who has no rights, and we will find the authorization error.



- Now provide overall read rights; build, read, and workspace rights for job for newly added users.

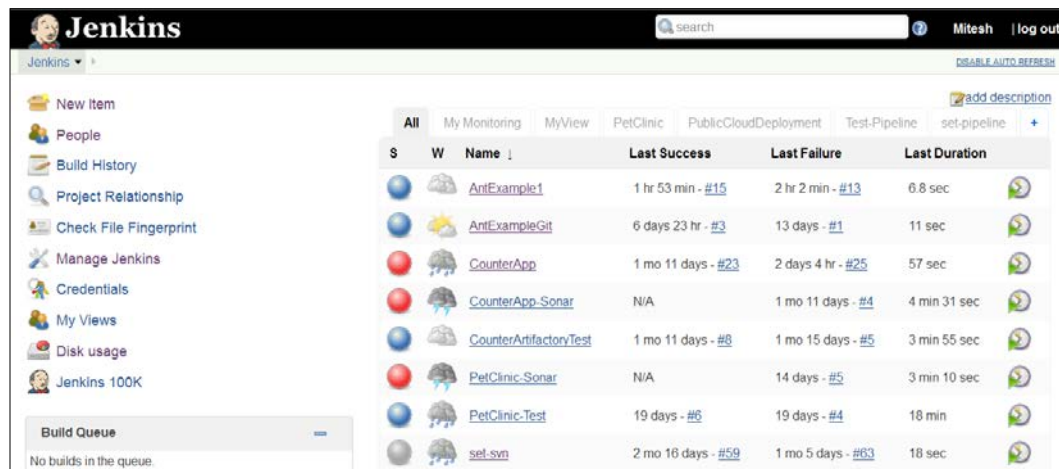
Authorization

☐ Anyone can do anything
☐ Legacy mode
☐ Logged-in users can do anything
☒ Matrix-based security

User/group	Overall				Credentials				Slave												
	Administer	Configure	Update	Center	Read	Run	Scripts	Upload	Plugins	Create	Delete	Manage	Domains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect
mites1983	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
mites1984	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☒	☒	☒	☒	☒	☒	☒

User/group to add:

- Log in with the newly added user and verify that we can see the dashboard. We can't see the **Manage Jenkins** link as we have provided those rights.



6. Click on any build job. The build link is available as we have given rights but the configure link is not available as rights were not given for it.

Project AntExample1

Project disk usage information + trend graph

Disk Usage: Workspace 61 MB (On slaves 61 MB, Non slave workspaces -), Builds 232 KB (Locked -), Job directory 31 MB

[Workspace](#)

[Recent Changes](#)

Build History

#	Timestamp	Size
#15	Jul 14, 2015 9:19 AM	13 KB
#14	Jul 14, 2015 9:18 AM	
#13	Jul 14, 2015 9:10 AM	
#12	Jul 12, 2015 7:28 AM	16 KB
#11	Jul 12, 2015 7:26 AM	21 KB

Permalinks

- Last build (#15), 1 hr 54 min ago
- Last stable build (#15), 1 hr 54 min ago
- Last successful build (#15), 1 hr 54 min ago
- Last failed build (#13), 2 hr 3 min ago
- Last unsuccessful build (#13), 2 hr 3 min ago

7. We can also set **Project-based Matrix Authorization Strategy**.

Authorization

☐ Anyone can do anything
☐ Legacy mode
☐ Logged-in users can do anything
☐ Matrix-based security
☒ Project-based Matrix Authorization Strategy

User/group	Overall	Credentials				Slave															
	Administer	Configure	Update	Center	Read	Run	Scripts	Upload	Plugins	Create	Delete	Manage	Domains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect
mitesh1983	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
mitesh1984	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

User/group to add:

8. Go to a specific build jobs' configuration and **Enable project-based security**.

Project name: AntExample1

Description:
 [Escaped HTML] [Preview](#)

☐ Discard Old Builds

☒ Enable project-based security

☐ Block inheritance of global authorization matrix

User/group	Credentials				Job								Run		SCM	
	Create	Delete	Manage	Domains	Update	View	Build	Cancel	Configure	Delete	Discover	Read	Workspace	Delete		Update
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
mitesh1983	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
mitesh1984	☐	☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

9. Assign rights to different users and log in with the specific username to verify whether authorization strategy is working or not.

Jenkins

Search: MiteshS / log out

People

Build History

Project Relationship

Check File Fingerprint

My Views

Disk usage

Jenkins 100K

Build Queue

No builds in the queue.

All My Monitoring MyView Test Pipeline

S	W	Name ↓	Last Success	Last Failure	Last Duration
		AntExample1	2 hr 16 min - #15	2 hr 25 min - #13	6.8 sec
		CounterApp	1 mo 11 days - #23	2 days 4 hr - #25	57 sec
		CounterApp-Sonar	N/A	1 mo 11 days - #4	4 min 31 sec
		CounterArtifactoryTest	1 mo 11 days - #8	1 mo 15 days - #5	3 min 55 sec

Icon: [S](#) [M](#) [L](#)

Legend RSS for all RSS for failures RSS for just latest builds

10. Verify the build details also, as shown in the following screenshot:

Jenkins search MiteshS | log out

Jenkins > AntExample1

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Subversion Polling Log](#)

Project AntExample1

Project disk usage information + trend graph

Disk Usage: Workspace 61 MB (On slaves 61 MB, Non slave workspaces -), Builds 232 KB (Locked -), Job directory 31 MB

[Recent Changes](#)

Build History

Build	Time	Size
#15	Jul 14, 2015 9:19 AM	13 KB
#14	Jul 14, 2015 9:18 AM	
#13	Jul 14, 2015 9:10 AM	
#12	Jul 12, 2015 7:28 AM	16 KB
#11	Jul 12, 2015 7:26 AM	21 KB
#10	Jul 12, 2015 7:14 AM	21 KB
#9	Jul 12, 2015 7:11 AM	17 KB
#8	Jul 12, 2015 7:09 AM	17 KB

Permalinks

- [Last build \(#15\), 2 hr 17 min ago](#)
- [Last stable build \(#15\), 2 hr 17 min ago](#)
- [Last successful build \(#15\), 2 hr 17 min ago](#)
- [Last failed build \(#13\), 2 hr 26 min ago](#)
- [Last unsuccessful build \(#13\), 2 hr 26 min ago](#)

We've covered basics of security configuration in Jenkins. Explore more on the other options as an exercise. In case, authorization is not correctly set, then it can be corrected by editing `config.xml`. Consider it as self-study.

Audit Trail Plugin – an overview and usage

Audit Trail Plugin keeps a log of users who performed particular Jenkins operations, such as configuring jobs. This plugin adds an **Audit Trail** section in the main Jenkins configuration page.

Install the **Audit Trail Plugin**.

Jenkins Mitesh | log out

Jenkins > Plugin Manager

[Back to Dashboard](#)

[Manage Jenkins](#)

Filter:

Updates Available **Installed** Advanced

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
☒	Audit Trail Keep a log of who performed particular Jenkins operations, such as configuring jobs.	2.1			Uninstall

In Jenkins configuration, configure **Loggers**, as shown in the following screenshot:

Audit Trail

Loggers

Log file

Log Location

Log File Size MB

Log File Count

Delete

Add Logger

Advanced...

Stop the Jenkins server and start it again. Run any build job and open log files to verify log records.

The screenshot shows a Jenkins console window titled "jenkins_logs.0.1 (/tmp) - gedit". The window has a menu bar (File, Edit, View, Search, Tools, Documents, Help) and a toolbar with icons for Open, Save, Undo, Redo, Cut, Copy, Paste, and other functions. The console output displays the following log entries:

```
May 24, 2015 12:07:09 PM job/CounterApp/ #15 Started by user Mitesh
May 24, 2015 12:07:26 PM job/CounterApp/com.mkymong$CounterWebApp/ #15 Started
May 24, 2015 12:07:46 PM CounterApp » CounterWebApp Maven Webapp #15 Started
on node Jenkins started at 2015-05-24T19:07:11Z completed in 0ms completed:
SUCCESS
May 24, 2015 12:07:47 PM CounterApp #15 Started by user Mitesh on node Jenkins
started at 2015-05-24T19:07:09Z completed in 37860ms completed: SUCCESS
```

At the bottom of the window, the status bar shows "Plain Text", "Tab Width: 8", "Ln 1, Col 1", and "INS".

To get more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/Audit+Trail+Plugin>.

Self-test questions

Q1. What are the different ways to make slave node online?

1. Launch an agent from the browser on slave
2. Run the `slave-agent.jnlp` command from the command line
3. Run `java -jar slave.jar`
4. All of the above

Q2. For what options does Jenkins monitoring provide charts?

1. CPU
2. Memory
3. System load average
4. HTTP response time
5. All of the above

Q3. What are the options for Security Realm in Jenkins?

1. Delegate to Servlet Container
2. Jenkins' own user database
3. LDAP
4. Unix user/group database
5. All of the above

Summary

Whatever good things we build end up building us. In this chapter, we covered concepts of master and slave nodes, how to monitor build jobs, and reporting of statistics with management features. We also understood how to secure Jenkins environment with authentication and authorization configurations by using role-based security. We saw how the audit trail plugin stores audit details in Jenkins.

In the next chapter, we will cover some important plugins that add a significant value to Jenkins. Let's enjoy the last journey before we say goodbye.

8

Beyond Basics of Jenkins – Leveraging "Must-have" Plugins

"Strength and growth come only through continuous effort and struggle."

- Napoleon Hill

In the last chapter, we covered management and monitoring along with security aspects in Jenkins. In security, we understood how authentication and authorization works. Now, it is time to recognize the value added by some important plugins.

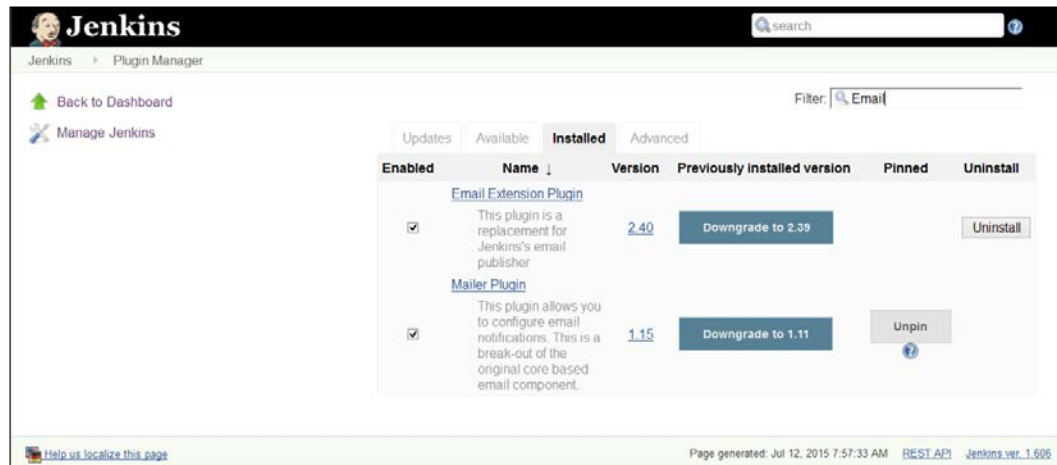
This chapter covers advanced usage of Jenkins, which is extremely useful in specific scenarios. Scenario-based usage of specific plugins that help development and operations teams are covered here for better utilization of Jenkins. Some of these plugins are extremely useful in the case of notifications scenario. The following are the main topics that we will cover in this chapter:

- Extended E-mail Plugin
- Workspace cleanup Plugin
- Pre-scm-buildstep Plugin
- Conditional BuildStep Plugin
- EnvInject Plugin
- Build Pipeline Plugin

Extended Email Plugin

Email-ext plugin extends functionality of e-mail notifications provided by Jenkins. It gives more customization in terms of conditions that cause mail notifications and content generation.

You can install this plugin from the Jenkin's dashboard.



Customization is available in three areas:

- Triggers: We can select the conditions that cause an e-mail notification to be sent
- Content: We can specify the content of each triggered email's subject and body; we can use default environment variables within content
- Recipients: We can specify who should receive an e-mail when it is triggered

In the Jenkins dashboard, click on **Manage Jenkins** and then click on **Configure System**. Go to the **Extended E-mail Notification** section and configure global email-ext properties that should match the settings for your SMTP mail server.

Extended E-mail Notification

SMTP server	smtp.gmail.com	?
Default user E-mail suffix		?
	Advanced...	
Default Content Type	HTML (text/html)	?
☐ Use List-ID Email Header		?
☒ Add 'Precedence: bulk' Email Header		?
Default Recipients	@gmail.com	?
Reply To List	@gmail.com	?
Emergency reroute		?
Excluded Recipients		?
Default Subject	\$PROJECT_NAME - Build # \$BUILD_NUMBER - \$BUILD_ST	?
Maximum Attachment Size		?

We can also customize the subject, maximum attachment size, default content, and so on.

Default Subject	\$PROJECT_NAME - Build # \$BUILD_NUMBER - \$BUILD_ST	?
Maximum Attachment Size		?
Default Content	\$PROJECT_NAME - Build # \$BUILD_NUMBER - \$BUILD_STATUS. Check console output at \$BUILD_URL to view the results.	?
Default Pre-send Script		?
Additional groovy classpath	Add	?
☐ Enable Debug Mode		?
☐ Enable Security		?
☐ Require Administrator for Template Testing		?
☐ Enable watching for jobs		?
	Default Triggers...	?
Content Token Reference		?
Save	Apply	

To configure Email-ext specific to build job, enable it in the project configuration page. Select the checkbox labeled **Editable Email Notification** in the **Post-build Actions**. Configure the comma- (or whitespace-) separated list of global recipients, subject, and content. In advanced configuration, we can configure pre-send script, triggers, email tokens, and so on.

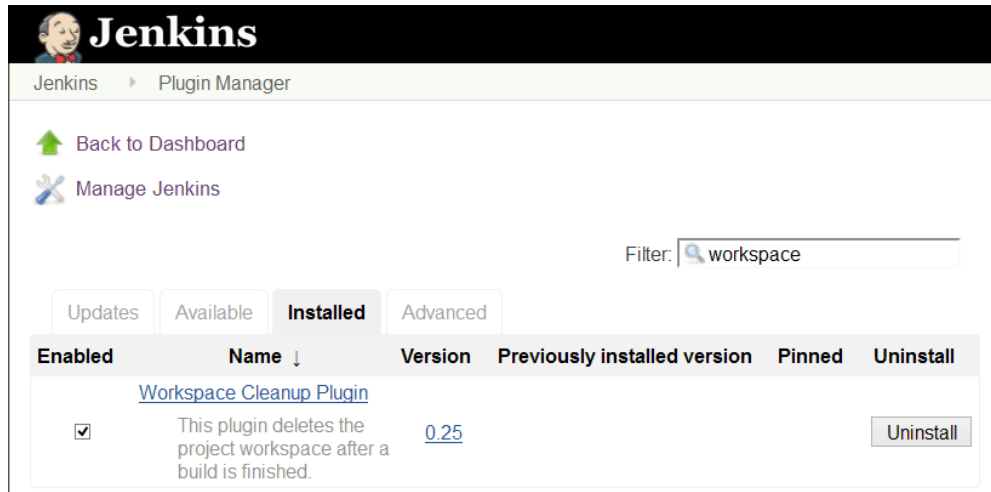
The screenshot shows the 'Editable Email Notification' configuration page in Jenkins. It includes a title bar with a grid icon and a help icon. Below the title is a checkbox for 'Disable Extended Email Publisher' with a help icon. A tooltip explains: 'Allows the user to disable the publisher, while maintaining the settings'. The main configuration area has several fields: 'Project Recipient List' (text input with '\$DEFAULT_RECIPIENTS' and a help icon), 'Project Reply-To List' (text input with '\$DEFAULT_REPLYTO' and a help icon), 'Content Type' (dropdown menu with 'Default Content Type' and a help icon), 'Default Subject' (text input with '\$DEFAULT_SUBJECT' and a help icon), 'Default Content' (text area with '\$DEFAULT_CONTENT' and a help icon), 'Attachments' (text input with a help icon), and 'Attach Build Log' (dropdown menu with 'Do Not Attach Build Log' and a help icon). A tooltip for the Attachments field states: 'Can use wildcards like 'module/dist/**/*.zip'. See the @includes of Ant files for the exact format. The base directory is the workspace'. At the bottom, there is a 'Content Token Reference' field and two buttons: 'Save' and 'Apply'.

The pre-send script feature allows us to write a script that can modify the `MimeMessage` object prior to sending the message. Triggers allow us to configure conditions that must be met to send an e-mail. The Email-ext plugin uses tokens to allow dynamic data to be inserted into the recipient list, e-mail subject line, or the body. For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/Email-ext+plugin>.

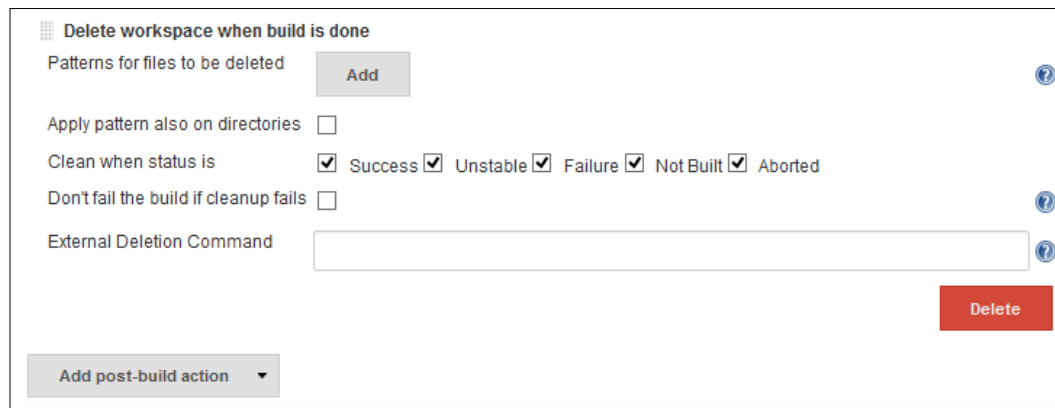
Workspace cleanup Plugin

The Workspace Cleanup plugin is used to delete the workspace from Jenkins before the build or when a build is finished and artifacts are saved. If we want to start a Jenkins build with a clean workspace or we want to clean a particular directory before each build, then we can effectively use this plugin. Different options are available for deleting the workspace.

You can install this plugin from the Jenkins dashboard.



We can apply patterns for files to be deleted based on the status of the build job. We can add post-build action for workspace deletion.

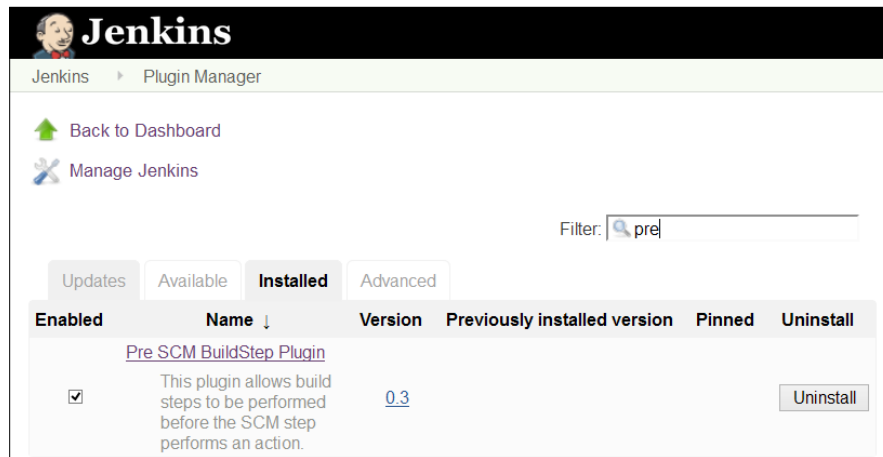


For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/Workspace+Cleanup+Plugin>.

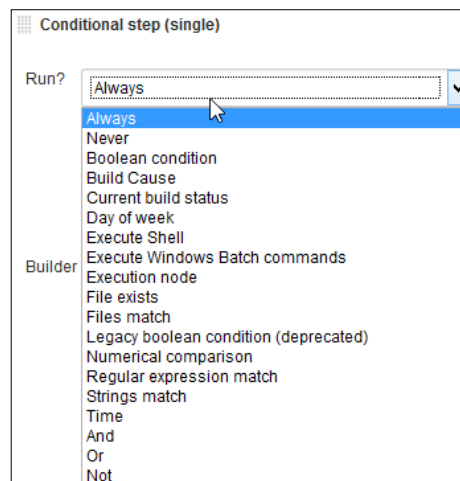
Pre-scm-buildstep Plugin

The Pre-scm-buildstep plugin allows a specific build step to run before SCM checkouts in case we need to perform any build step action on the workspace considering any special requirements such as adding a file with some settings for the SCM, executing some command to create some file, cleanup, or call other scripts that need to be run before checking out.

You can install this plugin from the Jenkins dashboard.



Select conditional steps from the list, as shown in the following screenshot:



Select the conditional steps based on requirement and provide a list of commands based on operating systems, as shown in the following screenshot:

The screenshot shows the 'Conditional step (single)' configuration window in Jenkins. It includes a 'Run?' dropdown set to 'Always', an 'Advanced...' button, a 'Builder' dropdown set to 'Execute shell', and a 'Command' text area containing the text 'declare -i n'. Below the command area is a link to 'See the list of available environment variables'. At the bottom are 'Save', 'Apply', and 'Delete' buttons.

Conditional step (single)

Run? Always

Advanced...

Builder Execute shell

Command declare -i n

See [the list of available environment variables](#)

Delete

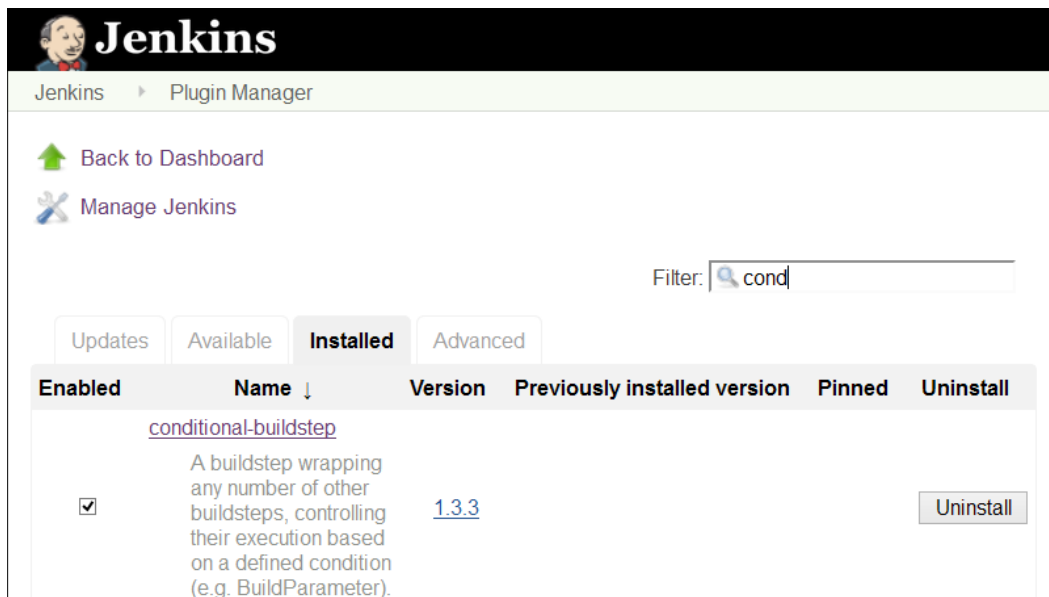
Save Apply

For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/pre-scm-buildstep>.

Conditional BuildStep Plugin

The Buildstep plugin allows us to wrap any number of other build steps, controlling their execution based on a defined condition.

You can install this plugin from the Jenkins' dashboard.



This plugin defines a few core run conditions such as:

- Always/Never: To disable a build step from the job configuration
- Boolean condition: To execute the step if a token expands to a representation of true
- Current status: To execute the build step if the current build status is within the configured/specific range
- File exists/Files match: To execute the step if a file exists, or matches a pattern
- Strings match: To execute the step if the two strings are same
- Numerical comparison: To execute the build step depending on the result of comparing two numbers
- Regular expression match: This provides a regular expression and a label, to execute the build step if the expression matches the label

- Time/Day of week: To execute the build job during a specified period of the day or day of the week
- And/Or/Not: Logical operations to enable the combining and sense inversion of run conditions
- Build cause: To execute the build step depending on the cause of the build, for example, triggered by timer, user, scm-change, and so on
- Script condition: Utilize shell script to decide whether a step should be skipped
- Windows Batch condition: Utilize windows batch to decide whether a step should be skipped

Select the **Conditional step (single)** from the **Add build step**.

Conditional step (single)

Run? File exists

File ec2.txt

Base directory Workspace

Advanced...

Builder Execute shell

Command declare -i n

[See the list of available environment variables](#)

Select the **Conditional steps (multiple)** from the **Add build step**. We can add multiple steps to condition in this conditional step.

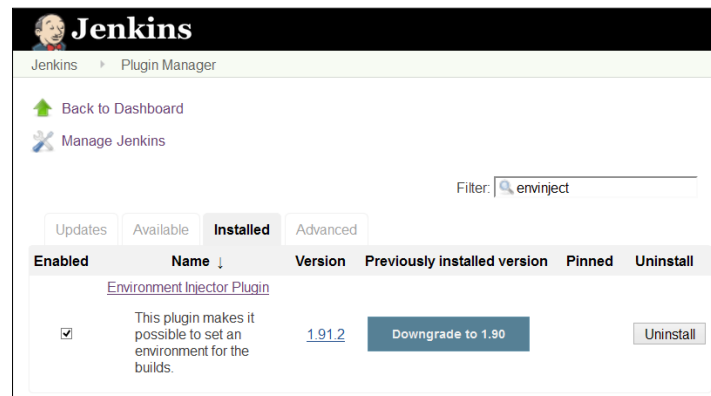
The screenshot shows the Jenkins configuration page for 'Conditional steps (multiple)'. At the top, there is a 'Run?' dropdown menu set to 'Always' with a help icon. Below this is an 'Advanced...' button. The main section is titled 'Steps to run if condition is met'. Inside this section, there is a step named 'Invoke OWASP Dependency-Check analysis'. Below the step name is a 'Path to scan' text input field with a help icon. To the right of the input field is another 'Advanced...' button. Below the input field is a red 'Delete' button. At the bottom left of the section is a button labeled 'Add step to condition' with a dropdown arrow.

For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/Conditional+BuildStep+Plugin>.

EnvInject Plugin

We know that different environments such as Dev, Test, and Production requires different configuration.

Install this plugin from the Jenkins dashboard.



The EnvInject plugin provides the facility to have an isolated environment for different build jobs. The EnvInject plugin injects environment variables at node startup, before or after a SCM checkout for a run, as a build step for a run, and so on. Select **Inject environment variables to the build process** specific to the build job.

☒ Inject environment variables to the build process

Properties File Path

Properties Content

Script File Path

Script Content

Evaluated Groovy script

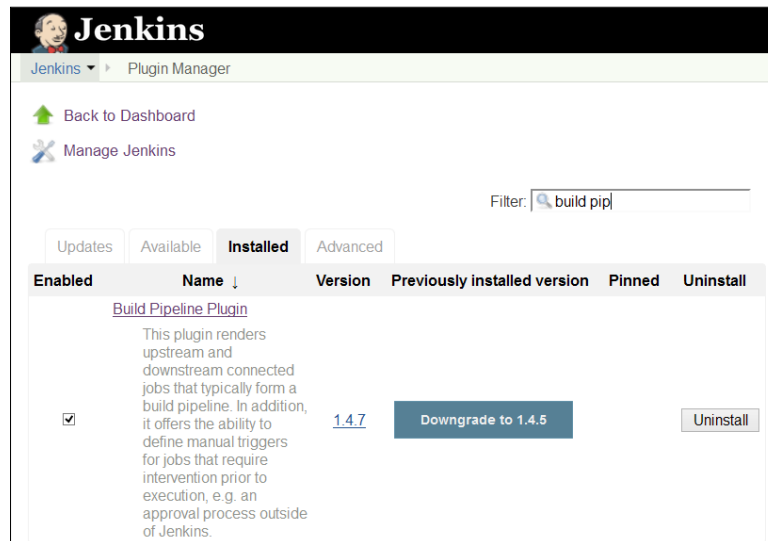
☐ Inject passwords to the build as environment variables

For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/EnvInject+Plugin>.

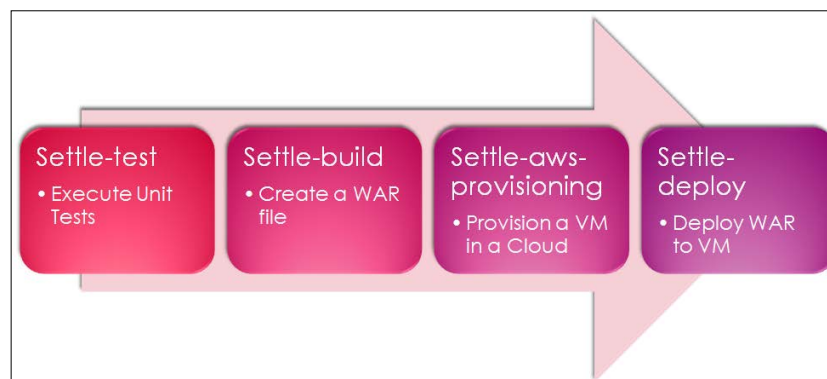
Build Pipeline Plugin

Continuous Integration has become a popular practice for application development. The Build Pipeline plugin provides a pipeline view of upstream and downstream connected jobs that typically form a build pipeline with the ability to define manual triggers or approval process. We can create a chain of jobs by orchestrating version promotion through different quality gates before we deploy it in production.

Install this plugin from the Jenkins dashboard.



We have already installed the Dashboard View plugin. We will create a pipeline for four build jobs. Let's assume we have four build jobs, as shown in the following diagram, where the objective of each build job is mentioned:



1. Create a new view and select **Build Pipeline View**.

Jenkins

View name: set-pipeline

- ☐ **Build Monitor View**
Shows a highly visible status of selected jobs. Ideal as an Extreme Feedback Device to be displayed on a screen on your office wall.
- ☐ **Build Pipeline View**
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.
- ☐ **Dashboard**
Customizable view that contains various portlets containing information about your job(s)
- ☐ **Delivery Pipeline View**
Shows one or more delivery pipeline instances.
- ☐ **Deployment Dashboard**
View for Deployment Dashboard. This dashboard integrates with an artifact repository to get all available versions of a specified artifact. It also integrates with your EC2 environments and gets the current status of your instances.
- ☐ **List View**
Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

Build Queue: No builds in the queue.

Build Executor Status

2. Provide a description and select the layout from the configuration on the build pipeline.
3. Select an initial job and the number of displayed builds and save the configuration.

Jenkins

Name: set-pipeline

Description:

Filter build queue: ☐

Filter build executors: ☐

Build Pipeline View Title:

Layout: Based on upstream/downstream relationship

Select Initial Job: settle-test

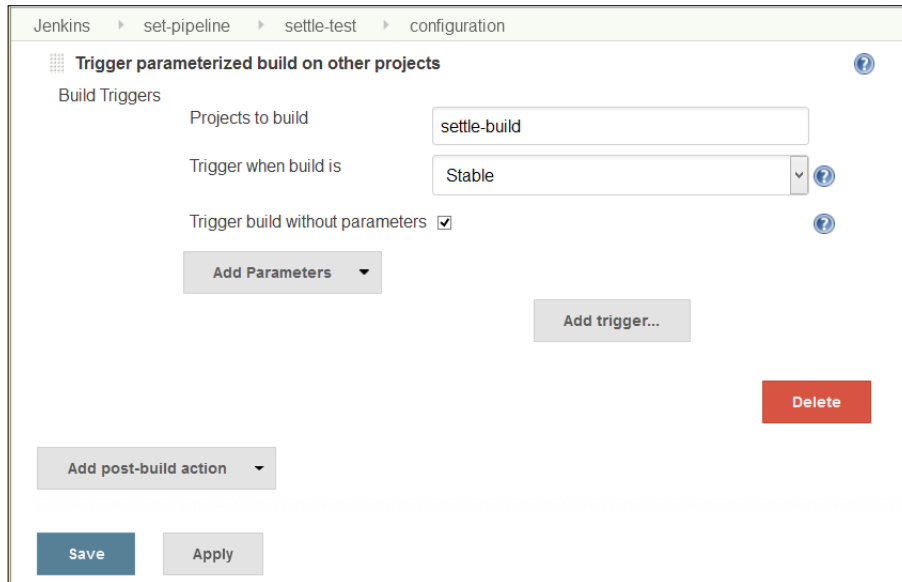
No Of Displayed Builds: 5

Restrict triggers to most recent successful builds: ☐ Yes ☒ No

Always allow manual trigger on pipeline steps: ☐ Yes ☒ No

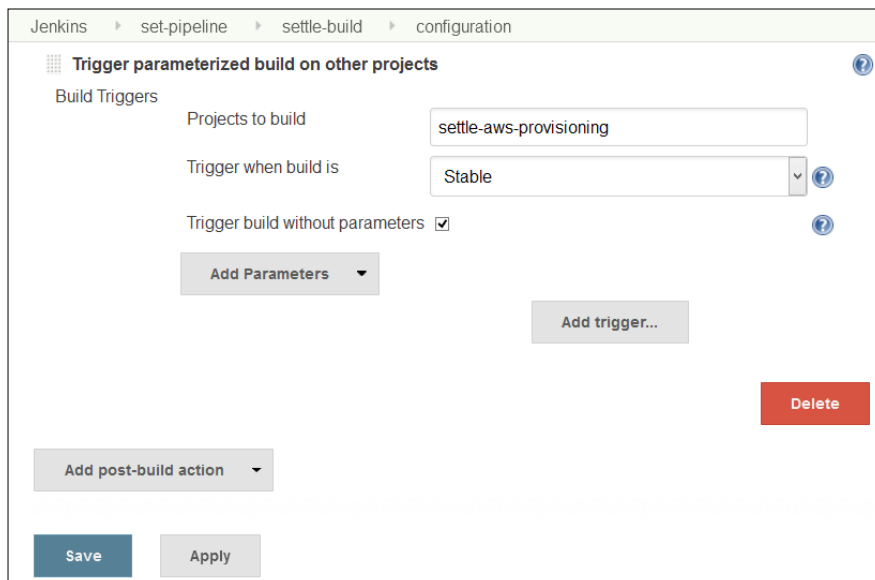
OK Apply

4. In a configuration of the build pipeline, select job to trigger parameterized build as `settle-build` job in **Post-build Actions**. It will be the first build job in the pipeline.



The screenshot shows the Jenkins configuration page for the 'settle-test' job. The breadcrumb trail is 'Jenkins > set-pipeline > settle-test > configuration'. The section title is 'Trigger parameterized build on other projects'. Under 'Build Triggers', the 'Projects to build' field contains 'settle-build'. The 'Trigger when build is' dropdown is set to 'Stable'. The 'Trigger build without parameters' checkbox is checked. There are buttons for 'Add Parameters', 'Add trigger...', and 'Delete'. At the bottom, there is an 'Add post-build action' dropdown and 'Save' and 'Apply' buttons.

5. In a `settle-build` job, trigger the parameterized build on the `settle-aws-provisioning` job in **Post-build Actions**.



The screenshot shows the Jenkins configuration page for the 'settle-build' job. The breadcrumb trail is 'Jenkins > set-pipeline > settle-build > configuration'. The section title is 'Trigger parameterized build on other projects'. Under 'Build Triggers', the 'Projects to build' field contains 'settle-aws-provisioning'. The 'Trigger when build is' dropdown is set to 'Stable'. The 'Trigger build without parameters' checkbox is checked. There are buttons for 'Add Parameters', 'Add trigger...', and 'Delete'. At the bottom, there is an 'Add post-build action' dropdown and 'Save' and 'Apply' buttons.

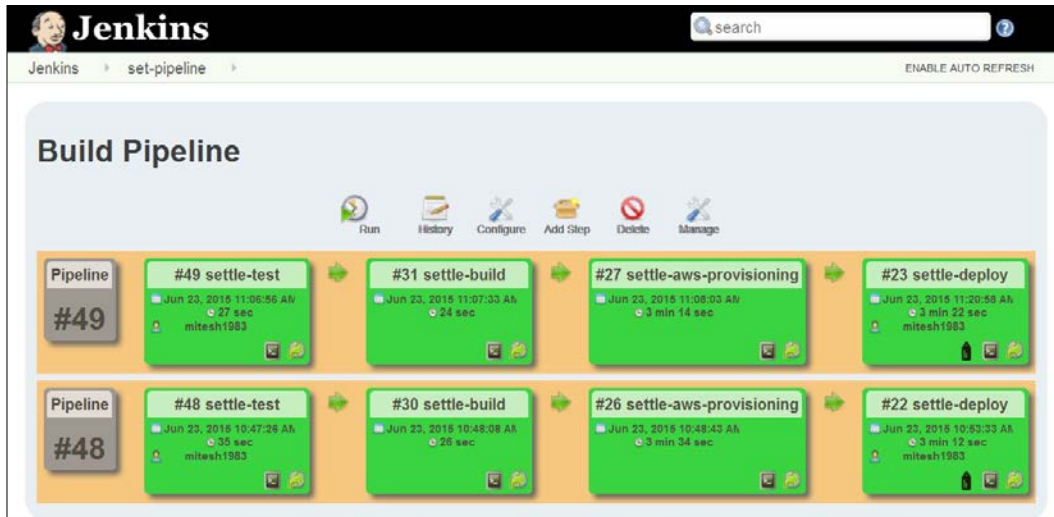
6. In a `settle-aws-provisioning` job, the manual build steps for a `settle-deploy` job in **Post-build Actions**.

The screenshot shows the Jenkins configuration page for the 'settle-aws-provisioning' job, specifically the 'Post-build Actions' section. The breadcrumb trail at the top is 'Jenkins > set-pipeline > settle-aws-provisioning > configuration'. The section title is 'Post-build Actions'. Below it, there is a grid icon followed by the text 'Build other projects (manual step)'. A text input field labeled 'Downstream Project Names' contains the value 'settle-deploy'. Below this field is a button labeled 'Add Parameters' with a dropdown arrow. Further down is another button labeled 'Add post-build action' with a dropdown arrow. At the bottom of the configuration area are two buttons: 'Save' (in a blue box) and 'Apply' (in a grey box).

7. In a `settle-aws-provisioning` job, trigger the parameterized build on the `settle-deploy` job in **Post-build Actions**. In the `settle-deploy` build job, we can write script or execute commands so that it can deploy war file to newly provisioned virtual machine in the cloud environment.

The screenshot shows the Jenkins configuration page for the 'settle-aws-provisioning' job, specifically the 'Copy artifacts from another project' section. The breadcrumb trail at the top is 'Jenkins > set-pipeline > settle-aws-provisioning > configuration'. The section title is 'Copy artifacts from another project'. Below it, there are several input fields: 'Project name' with the value 'settle-build', 'Which build' with the value 'Latest successful build', and a checkbox for 'Stable build only' which is currently unchecked. Below these are three more input fields: 'Artifacts to copy' with the value '**/*.war', 'Artifacts not to copy' (empty), 'Target directory' (empty), and 'Parameter filters' (empty). At the bottom of the configuration area are two buttons: 'Save' (in a blue box) and 'Apply' (in a grey box).

8. Go to the dashboard view, which we created earlier, and verify the pipeline created after our configuration in build jobs in the previous section. The new build pipeline will be created as shown in the following diagram:



For more details, visit <https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin>.

Self-test questions

Q1. In which areas does the Extended E-mail plugin provide customization?

1. Triggers
2. Content
3. Recipients
4. All of the above

Q2. The Workspace cleanup plugin provides an option to clean the workspace when status of build is:

1. Success
2. Unstable
3. Failure
4. Not Built
5. Aborted
6. All of the above

Summary

We learned how to use some important plugins to aid the existing features of Jenkins to address specific requirements. We covered all basic usage of Jenkins, including installing runtime environment, creating build jobs, using Jenkins on Cloud, monitoring, management, security, and additional plugins. For the scope of this book, this seems sufficient. Next step is about provisioning resources dynamically in Cloud environment to achieve end to end automation in the DevOps journey.

If you want a happy ending, that depends of course on where you stop your story. We certainly know where to stop ours!

Index

A

access control

managing 129-132

Android Lint plugin

about 113

URL 113

Ant

configuring 39

configuring, in Jenkins 40, 41

installing 38

URL 38

Apache jclouds case study

about 96

benefits 97

challenge 96

reference 97

solution 96

Audit Trail Plugin

about 137

installing 137, 138

URL 138

authorization

managing 129-132

B

build

executing, with test cases 62-64

build job

creating in Jenkins, Git used 44-47

build job, Java application

configuring, Ant used 50-57

configuring, Maven used 59-61

creating, Ant used 50-52

creating, Maven used 58

Build Monitor plugin

about 127

build monitoring with 127, 128

URL 128

Build Pipeline plugin

about 152

installing 152-155

URL 156

C

case studies, from CloudBees

about 96

Apache jclouds 96

Global Bank 97

Service-Flow 98

CCM plugin

about 113

URL 113

Checkpoints plugin

about 94

URL 95

Checkstyle plugin

about 110

URL 110

CloudBees

about 84

Jenkins, exploring in 84-94

URL 84

CloudBees Enterprise plugins

Checkpoints plugin 94

High Availability plugin 95

overview 94

Role-based Access Control (RBAC)
plugin 95

- VMware ESXi/vSphere Auto-Scaling
 - plugin 95
- Workflow plugin 94
- Compiler Warnings plugin**
 - about 111-113
 - URL 111
- Conditional BuildStep plugin**
 - about 148
 - installing 148-150
 - URL 150
- continuous delivery** 68
- continuous deployment** 68
- continuous integration (CI) tools** 1
- cron expression**
 - reference 3

D

- dashboard, Jenkins** 10
- Dashboard View Plugin**
 - about 50
 - installing 50-52
 - URL 50
- deployment pipeline, Jenkins** 15
- Deploy plugin**
 - about 70
 - URL 70
 - using 70
- DEV@cloud** 84
- disk usage**
 - managing 125, 126
- Disk Usage plugin**
 - about 125
 - URL 126
- DRY plugin**
 - about 113
 - URL 113

E

- Eclipse**
 - integrating, with code repositories 36-38
- e-mail notifications, based on build status**
 - sending 114
- EnvInject plugin**
 - about 150
 - installing 150, 151
 - URL 151

- Extended Email plugin**
 - about 142
 - configuring 144
 - installing 142, 143
 - URL 144

F

- FindBugs plugin**
 - about 111
 - URL 111

G

- Git**
 - about 41
 - configuring 41
 - installing 41-43
- Global Bank case study**
 - about 97
 - benefits 98
 - challenge 97
 - reference 98
 - solution 97

H

- High Availability plugin**
 - about 95
 - URL 95

I

- installation**
 - Jenkins 4
 - Jenkins, as web application 9, 10
 - Jenkins on CentOS 7, 8
 - Jenkins on Windows 4-6

J

- Java**
 - environment variables, configuring 22
 - installing 20-22
- JavaMelody**
 - Jenkins monitoring with 123-125
- JDK**
 - configuring, in Jenkins 40

Jenkins

- about 1, 2
- build 20
- CI 2
- configuration settings, changing 12-14
- dashboard 10
- deployment pipeline 15
- exploring, in CloudBees 84-94
- exploring, in OpenShift PaaS 80-83
- features 3
- installing 4
- installing, as web application 9, 10
- installing, on CentOS 7, 8
- installing, on Windows 4-6
- monitoring, with JavaMelody 123-125
- plugins 2
- requisites 20
- URL 4

M

master nodes

- managing 118-122

Maven

- configuring, in Jenkins 41
- installing 39
- URL 39

O

OpenShift Online

- about 80
- URL 80

OpenShift PaaS

- Jenkins, exploring in 80-83

OWASP Dependency-Check Plugin

- about 114
- URL 114

P

plugins

- references 3

PMD plugin

- about 113
- URL 113

Pre-scm-buildstep plugin

- about 146

- installing 146

- URL 147

project-based security

- maintaining 133-137

R

Role-based Access Control (RBAC) plugin

- about 95
- features 95
- URL 95

roles

- maintaining 133-137

S

Service-Flow case study

- about 98
- benefits 98
- challenge 98
- solution 98

slave nodes

- managing 118-122

SonarQube

- about 101
- integrating with 101-110
- URL 102

Static Code Analysis plugins

- about 110
- Android Lint plugin 113
- CCM plugin 113
- Checkstyle plugin 110
- Compiler Warnings plugin 111-113
- DRY plugin 113
- exploring 110
- FindBugs plugin 111
- OWASP Dependency-Check Plugin 114
- PMD plugin 113
- Task Scanner plugin 113

SVN

- configuring 24-26
- directory, importing to 26
- installing, on CentOS 23
- operations 26
- source code, checking out 27

SVN client

- URL 32

T

Task Scanner plugin

about 113

URL 113

Tomcat

about 68

installing 68, 69

V

VisualSVN server

installing, on Windows 28-36

URL 28

VMware ESXi/vSphere Auto-Scaling plugin

about 95

URL 96

W

war file

deploying, from Jenkins to Tomcat 70-76

Workflow plugin

about 94

URL 94

Workspace Cleanup plugin

about 144

installing 145

URL 145



Thank you for buying Jenkins Essentials

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at www.packtpub.com.

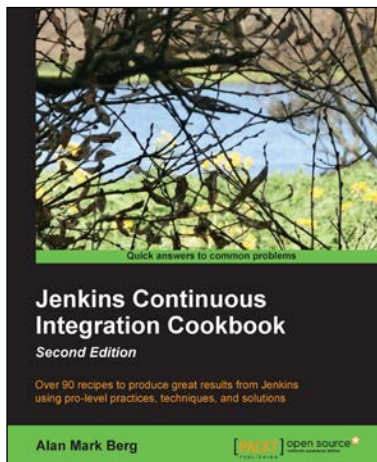
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around open source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each open source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



Jenkins Continuous Integration Cookbook

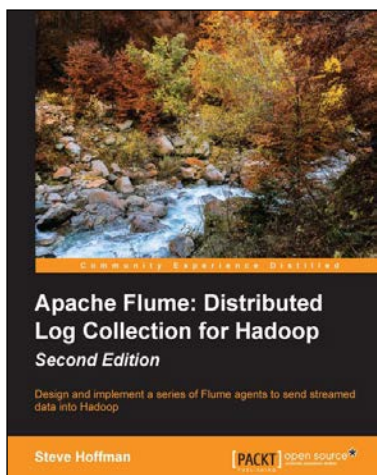
Second Edition

ISBN: 978-1-78439-008-2

Paperback: 408 pages

Over 90 recipes to produce great result from Jenkins using pro-level practices, techniques, and solutions

1. Explore the use of more than 40 best-of-breed plug-ins for improving efficiency.
2. Secure and maintain Jenkins by integrating it with LDAP and CAS, which is a Single Sign-on solution.
3. Step-by-step, easy-to-use instructions to optimize the existing features of Jenkins using the complete set of plug-ins that Jenkins offers.



Apache Flume: Distributed Log Collection for Hadoop

Second Edition

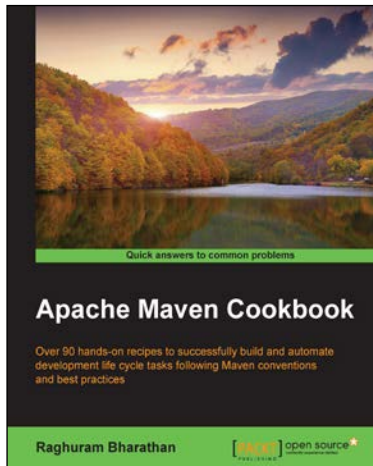
ISBN: 978-1-78439-217-8

Paperback: 178 pages

Design and implement a series of Flume agents to send streamed data into Hadoop

1. Construct a series of Flume agents using the Apache Flume service to efficiently collect, aggregate, and move large amounts of event data.
2. Configure failover paths and load balancing to remove single points of failure.
3. Use this step-by-step guide to stream logs from application servers to Hadoop's HDFS.

Please check www.PacktPub.com for information on our titles



Apache Maven Cookbook

ISBN: 978-1-78528-612-4 Paperback: 272 pages

Over 90 hands-on recipes to successfully build and automate development life cycle tasks following Maven conventions and best practices

1. Understand the features of Apache Maven that makes it a powerful tool for build automation.
2. Full of real-world scenarios covering multi-module builds and best practices to make the most out of Maven projects.
3. A step-by-step tutorial guide full of pragmatic examples.



Learning Force.com Application Development

ISBN: 978-1-78217-279-6 Paperback: 406 pages

Use the Force.com platform to design and develop real-world, cutting-edge cloud applications

1. Design, build, and customize real-world applications on the Force.com platform.
2. Reach out to users through public websites and ensure that your Force.com application becomes popular.
3. Discover the tools that will help you develop and deploy your application.

Please check www.PacktPub.com for information on our titles